FP Complete

Hedera Token Service

AUDIT REPORT

Customer Name:	Hedera Hashgraph
Version:	1.0
Submitted to:	Mehernosh Mody

Table of Contents

Prologue

This report is designated for **external distribution**, in accordance with the disclaimer below.

Disclaimer

This report is authorized for external distribution. This report is presented without warranty or guarantee of any type.

This report touches aspects of both the code itself and the architecture. Information for the architecture is gleaned from the whitepaper, website, and discussions with Swirlds members.

THIS REPORT IS UNCORRECTED AND INCOMPLETE, IS BASED ON INCOMPLETE INFORMATION, AND IS PRESENTED WITHOUT WARRANTY OR GUARANTY OF ANY TYPE.

This report is a copy of a work in progress. It lists the most salient concerns that have so far become apparent to FP Complete after a partial inspection of the engineering work. The inspection is ongoing, so further concerns are likely to arise. Corrections, such as the cancellation of incorrectly reported issues, may also arise. Therefore, FP Complete advises against making any business decision or other decision based on this report.

FP COMPLETE DOES NOT RECOMMEND FOR OR AGAINST THE USE OF ANY WORK OR SUPPLIER REFERENCED IN THIS REPORT.

This report focuses on the technical implementation as provided by the project's implementors, based on information provided by them, and is not meant to assess the concept, mathematical validity, or business validity of the project. This report does not make any assessment of the implementation or the project regarding financial viability, nor suitability for any purpose.

While this assessment when complete might be described as an "audit," no official standard exists for an audit of this nature. The word "audit" does not imply compliance with an accounting standard or other standard and is used informally here. FP Complete has not been given access to nor reviewed all aspects of the project and the engineering decision process underlying all the work. This report likely contains errors due to incomplete information as well as simple misunderstanding. This report may include references to problems that do not in fact exist. Meanwhile, the work referenced may or may not contain undetected or unreported problems. FP Complete has not had independent and unfettered access to all the relevant materials. Nor has a "whistleblower" or other process been provided such that any known problem could be reported and included herein.

Some technical decisions in the engineering work were made due to historic reasons, time constraints, budget constraints, or other constraints. Therefore, the presence of a concern or "flag" in this report does not imply improper conduct or lack of skill by the implementer or manager or any party.

NO ATTEMPT IS MADE OR IMPLIED TO JUDGE ANY PERSON, TEAM, COMPANY, OR OTHER PARTY.

Source Material

For this report, the FP Complete team has reviewed the Hedera Token Service, which is part of the Hedera code base. The source materials consist of:

- The public documentation available at https://docs.hedera.com/guides/
- The hedera-services repository at https://github.com/hashgraph/hedera-services up to commit SHA feab04af3f4e8c18915fb42ca2cb3395c43bd885.

Furthermore, the audit explicitly excluded:

- Code that is not related to the Hedera Token Service (HTS)
- Test code
- Automatically generated code
- DevOps, infrastructure, and network architecture
- Technical leadership
- Applicability for business use-cases
- Hiring

Considerations

After fixes, previously identified issues have been resolved. For brevity, those issues are not included in this report.

Legend

This report classes findings into two categories:

- Red flag Confirmed issue which should be addressed immediately
- Yellow flag Potential problem without a clear or immediate exploit

This report does not include past findings of the repository hedera-services raised in previous audits. Instead, we list unresolved and partially resolved flags that comment about production code of the Hedera Token Service.

Executive Summary

This report contains 0 red flag issues and 3 yellow flag issues. These issues remain partially resolved or unresolved at the time of writing.

Unresolved and Partially Resolved Findings

This section lists all partially resolved and unresolved findings.

Yellow flag: Misuse of builder design pattern

Reproducible in commit: feab04af3f4e8c18915fb42ca2cb3395c43bd885

Impact

The described issue decreases the maintainability and benefits the introduction of future bugs.

Situation

All *Usage classes follow more or less this pattern: the method newEstimate() creates an initial instance, other methods might exist to update it, and the get() method returns a FeeData value for this usage instance. For example, the class TokenUpdateResourceUsage uses this pattern.

Issue

In the following, we list multiple concerns with the chosen implementation

1. Repeated calls to get() yield different FeeData:

   Calling get() modifies the the usage instance, which is confusing as the term "get" refers to a read-only operation that does not modify data. As a result, two consecutive calls to get() return two different FeeData objects.

2. Partially idempotent API:

   The QueryUsage class, which also reminds of the builder pattern, contains the methods updateRb and updateTb which both add to the underlying value instead of replacing it. Based on the builder pattern and the method names themselves, both methods are expected to be idempotent.

   These methods are used in *Usage classes, e.g. in TokenGetInfoUsage. Due to the nature of updateRB and updateTb, the methods givenCurrentName, givenCurrentSymbol, and givenCurrentlyUsingAutoRenewAccount are not idempotent. Calling these methods more than once, accidentially or not, will modify the underlying data on each call. This results in an inconsistent and possibly dangerous to use API.

3. Negative values of Rbs are possible

   The method TokenAssociatedUsage::get is defined as:

```
public FeeData get() {
  var op = this.op.getTokenAssociate();
  addAccountBpt();
  op.getTokensList().forEach(t -> addAccountBpt());
  novelRelsLasting(op.getTokensCount(), ESTIMATOR_UTILS.relativeLifetime(this.op, currentExpiry
));
```

```
  return usageEstimator.get();
}
```

In case a call to this method is not preceded by a call to givenCurrentExpiry, Rbs is set to a negative value, which seems undesired. Since its initial value is zero, the method relativeLifetime() function will return a negative value. A related cause can be the variable expiry, which is optional but considered mandatory by the calculation.

Added documentation about the call sequence, or a fix of the code, are advisable.

4. Unnecessary multi-step builders:

   Many *Usage implementations do not take parameters which would customize the FeeData before it is returned from get. In these implementations, a call to newEstimate is effectively always directly followed by a call to get, which makes the builder pattern obsolete. Instead, a single method creating FeeData could be favored to avoid unnecessary intermediate steps. For example, this applies to TokenRevokeKycUsage.java, TokenMintUsage.java, and TokenAssociateResourceUsage.java.

5. Complexity of nested builders:

   Nesting builders seems to over-complicate the code. For example, the class TokenAssociateUsage is a builder that extends from TxnUsageEstimator, which is also a builder, which itself uses the builder UsageEstimate in its get method.

6. Unclear intentions and suspicious code:

   There are places in the code that look like this:

```
public FeeData get() {
  addAccountBpt();
  addAccountBpt();
  return usageEstimator.get();
}
```

   In this extract from TokenFreezeUsage.java#L45, addAccountBpt is called twice so that usageEstimator is incremented twice with AMOUNT_REPR_BYTES. When reading this code, it is unclear if the double-addition is intended or is a copy-paste mistake. More documentation or a less ambiguous implementation would clarify the situation. Other examples of this issue are: TokenUnfreezeUsage.java#L45, TokenRevokeKycUsage.java#L45, and TokenWipeUsage.java#L45.

Yellow flag: Dearth of source code documentation

Reproducible in commit: feab04af3f4e8c18915fb42ca2cb3395c43bd885

Impact

This may lead to difficulty in establishing context around the undocumented code for further development and maintenance.

Issues

Using the PMD tool, by its CLI, we generated a report for missing Javadocs:

```
❯ pmd -d hapi-fees -f text -R ./javadoc-ruleset.xml > comments-required.txt
```

where, javadoc-ruleset.xml is the following:

```
<?xml version="1.0"?>

<ruleset name="Missing Javadoc"
  xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 https://pmd.sourceforge.io/ruleset_2_0_0
.xsd">

  <description>
    Missing Javadoc
  </description>

  <rule ref="category/java/documentation.xml/CommentRequired">
    <properties>
      <property name="methodWithOverrideCommentRequirement" value="Ignored" />
      <property name="accessorCommentRequirement" value="Ignored" />
      <property name="classCommentRequirement" value="Required" />
      <property name="fieldCommentRequirement" value="Required" />
      <property name="publicMethodCommentRequirement" value="Required" />
      <property name="protectedMethodCommentRequirement" value="Required" />
      <property name="enumCommentRequirement" value="Required" />
      <property name="serialVersionUIDCommentRequired" value="Ignored" />
      <property name="serialPersistentFieldsCommentRequired" value="Ignored" />
    </properties>
  </rule>

</ruleset>
```

This reports 165 violations of the rule CommentRequired with test sources excluded. This report can be found in the asset comments-required-hapi-fees.txt

When we run this analysis on a larger, project-wide scope (hedera-services), 20073 violations are reported (test sources excluded). This report can be found in the asset comments-required-hedera-services.txt

This number is directly indicative of significant elements of source code, such as, definitions of classes, methods, functions, arguments, variables and others, not being accompanied by any documentation. This can degrade the quality of further development and maintenance of the project.

Recommendation

1. Decrease the number of violations reported by PMD by documenting more of the exiting source code.
2. Proactively document source code alongside new development.

Yellow flag: Inconsistencies in transaction validations

Reproducible in commit: feab04af3f4e8c18915fb42ca2cb3395c43bd885

Impact

The described issue decreases the maintainability and can slightly decrease the performance.

Issues

We collect a few issues that concern token transactions, in particular their transition logic and validations.

1. The TokenCreateTransitionLogic checks if the expiry is after the transaction consensus time by performing an inline check. In contrast, TopicUpdateTransitionLogic, L.154 and CryptoUpdateTransitionLogic, L.149 use the OptionValidator::isValidExpiry method for the identical check. The utility method should be used consistently.

2. Validation logic is spread across *TransitionLogic classes and the HederaTokenStore. For instance, the check for a missing token (outcome: MISSING_TOKEN) is duplicated in TokenUpdateTransitionLogic::transitionFor and HederaTokenStore::update.

3. The TokenCreateTransitionLogic validates both expiry and autorenew as part of the pre-check in the validate() method. Neither of these properties is validated in TokenUpdateTransitionLogic::validate(). However, the TokenUpdate logic relies on HederaTokenStore::update to check if a new expiry value is set to a time that is later than the current token's expiry.

4. The previous point also introduces the problem of different times of validation: TokenCreateTransitionLogic performs validation as pre-checks in the validate() method, while TokenUpdateTransitionLogic results in the validation being performed at post-consensus execution time.