FYEO

Security Review of Xahaud - Codebase for Xahau

XRPL Labs

October 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

EXECUTIVE SUMMARY

OVERVIEW

XRPL Labs engaged FYEO Inc. to perform a Security Review of Xahaud - Codebase for Xahau.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation. **During the audit, the XRPL Labs team adequately remediated all issues listed in this report.**

The assessment was conducted remotely by the FYEO Security Team. Testing took place on October 01 - October 27, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

KEY FINDINGS

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-XAH01-01 – Import transaction does not properly check signer list
- FYEO-XAH01-02 – Inconsistent check of isFieldPresent(sfTransactionResult)
- FYEO-XAH01-03 – Values that would have caused overflows are ignored
- FYEO-XAH01-04 – syntaxCheckXPOP checks validation unl in for loop making fields optional
- FYEO-XAH01-05 – xpop_slot may write data in the same slot
- FYEO-XAH01-06 – Comment indicates functionality different from what is implemented
- FYEO-XAH01-07 – Only the last symbol in memory is checked to be not 0
- FYEO-XAH01-08 – Hard to read functions impede maintainability
- FYEO-XAH01-09 – Inconsistent check of pair returned by getInnerTxn

- FYEO-XAH01-10 – Json value type is checked in assert before processing
- FYEO-XAH01-11 – Make sure that featureExpandedSignerList is enabled in Import::doSignerList()
- FYEO-XAH01-12 – The determineOperation function returns success for unknown operations
- FYEO-XAH01-13 – The function syntaxCheckXPOP does not verify that ledger::index exists
- FYEO-XAH01-14 – While loop that does not iterate

Based on our review process, we conclude that the reviewed code implements the documented functionality.

SCOPE AND RULES OF ENGAGEMENT

The FYEO Review Team performed a Security Review of Xahaud - Codebase for Xahau. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/Xahau/xahaud with the commit hash dev.

Files included in the code review

```
xahaud/
├── hook/
│   ├── error.h
│   ├── extern.h
│   ├── govern.c
│   ├── hookapi.h
│   ├── macro.h
│   ├── mint.c
│   ├── nftoken.c
│   ├── reward.c
│   ├── sfcodes.h
│   └── tts.h
└── src/
    └── ripple/
        └── app/
            ├── hook/
            │   ├── impl/
            │   │   └── applyHook.cpp
            │   ├── Enum.h
            │   ├── Guard.h
            │   ├── Macro.h
            │   ├── Misc.h
            │   ├── applyHook.h
```

Files included in the code review

```
    │   ├── guard_checker.cpp
    │   └── xahau.h
    ├── misc/
    │   ├── detail/
    │   │   ├── impl/
    │   │   │   └── WorkSSL.cpp
    │   │   ├── Work.h
    │   │   ├── WorkBase.h
    │   │   ├── WorkFile.h
    │   │   ├── WorkPlain.h
    │   │   └── WorkSSL.h
    │   ├── impl/
    │   │   ├── AccountTxPaging.cpp
    │   │   ├── AccountTxPaging.h
    │   │   ├── AmendmentTable.cpp
    │   │   ├── LoadFeeTrack.cpp
    │   │   ├── Manifest.cpp
    │   │   ├── Transaction.cpp
    │   │   ├── TxQ.cpp
    │   │   ├── ValidatorKeys.cpp
    │   │   ├── ValidatorList.cpp
    │   │   └── ValidatorSite.cpp
    │   ├── AmendmentTable.h
    │   ├── CanonicalTXSet.cpp
    │   ├── CanonicalTXSet.h
    │   ├── FeeVote.h
    │   ├── FeeVoteImpl.cpp
    │   ├── HashRouter.cpp
    │   ├── HashRouter.h
    │   ├── LoadFeeTrack.h
    │   ├── Manifest.h
    │   ├── NegativeUNLVote.cpp
    │   ├── NegativeUNLVote.h
    │   ├── NetworkOPs.cpp
    │   ├── NetworkOPs.h
    │   ├── SHAMapStore.h
    │   ├── SHAMapStoreImp.cpp
    │   ├── SHAMapStoreImp.h
    │   ├── Transaction.h
    │   ├── TxQ.h
    │   ├── ValidatorKeys.h
    │   ├── ValidatorList.h
    │   └── ValidatorSite.h
```

Files included in the code review

```
    └── tx/
        ├── impl/
        │   ├── details/
        │   │   ├── NFTokenUtils.cpp
        │   │   └── NFTokenUtils.h
        │   ├── ApplyContext.cpp
        │   ├── ApplyContext.h
        │   ├── BookTip.cpp
        │   ├── BookTip.h
        │   ├── CancelCheck.cpp
        │   ├── CancelCheck.h
        │   ├── CancelOffer.cpp
        │   ├── CancelOffer.h
        │   ├── CashCheck.cpp
        │   ├── CashCheck.h
        │   ├── Change.cpp
        │   ├── Change.h
        │   ├── ClaimReward.cpp
        │   ├── ClaimReward.h
        │   ├── CreateCheck.cpp
        │   ├── CreateCheck.h
        │   ├── CreateOffer.cpp
        │   ├── CreateOffer.h
        │   ├── CreateTicket.cpp
        │   ├── CreateTicket.h
        │   ├── DeleteAccount.cpp
        │   ├── DeleteAccount.h
        │   ├── DepositPreauth.cpp
        │   ├── DepositPreauth.h
        │   ├── Escrow.cpp
        │   ├── Escrow.h
        │   ├── GenesisMint.cpp
        │   ├── GenesisMint.h
        │   ├── Import.cpp
        │   ├── Import.h
        │   ├── InvariantCheck.cpp
        │   ├── InvariantCheck.h
        │   ├── Invoke.cpp
        │   ├── Invoke.h
        │   ├── NFTokenAcceptOffer.cpp
        │   ├── NFTokenAcceptOffer.h
        │   ├── NFTokenBurn.cpp
        │   ├── NFTokenBurn.h
```

Files included in the code review
```
│       ├── NFTokenCancelOffer.cpp
│       ├── NFTokenCancelOffer.h
│       ├── NFTokenCreateOffer.cpp
│       ├── NFTokenCreateOffer.h
│       ├── NFTokenMint.cpp
│       ├── NFTokenMint.h
│       ├── Offer.h
│       ├── OfferStream.cpp
│       ├── OfferStream.h
│       ├── PayChan.cpp
│       ├── PayChan.h
│       ├── Payment.cpp
│       ├── Payment.h
│       ├── SetAccount.cpp
│       ├── SetAccount.h
│       ├── SetHook.cpp
│       ├── SetHook.h
│       ├── SetRegularKey.cpp
│       ├── SetRegularKey.h
│       ├── SetSignerList.cpp
│       ├── SetSignerList.h
│       ├── SetTrust.cpp
│       ├── SetTrust.h
│       ├── SignerEntries.cpp
│       ├── SignerEntries.h
│       ├── Taker.cpp
│       ├── Taker.h
│       ├── Transactor.cpp
│       ├── Transactor.h
│       ├── URIToken.cpp
│       ├── URIToken.h
│       ├── XahauGenesis.h
│       ├── apply.cpp
│       └── applySteps.cpp
├── apply.h
└── applySteps.h
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Review of Xahaud - Codebase for Xahau, we discovered:

- 1 finding with HIGH severity rating.
- 4 findings with MEDIUM severity rating.
- 2 findings with LOW severity rating.
- 7 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

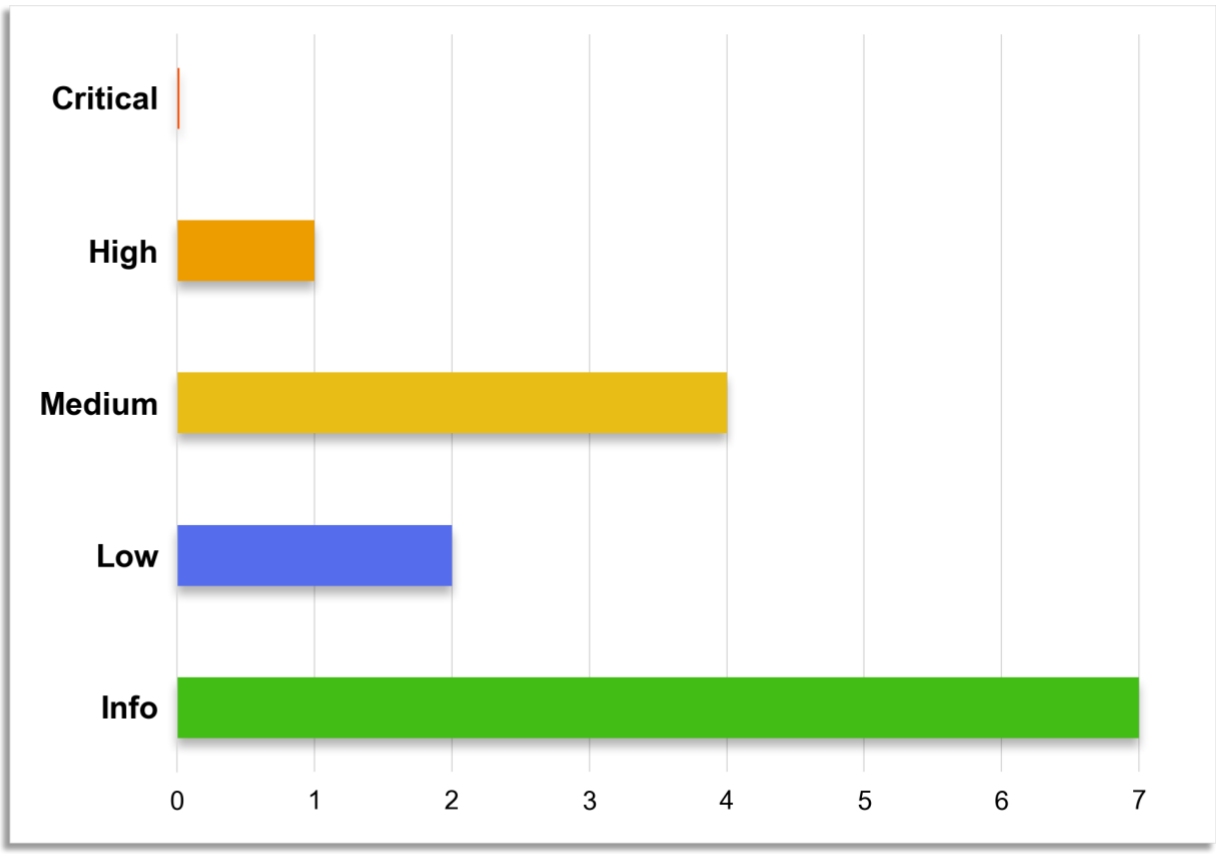


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-XAH01-01	**High**	Import transaction does not properly check signer list
FYEO-XAH01-02	**Medium**	Inconsistent check of isFieldPresent(sfTransactionResult)
FYEO-XAH01-03	**Medium**	Values that would have caused overflows are ignored
FYEO-XAH01-04	**Medium**	syntaxCheckXPOP checks validation unl in for loop making fields optional
FYEO-XAH01-05	**Medium**	xpop_slot may write data in the same slot
FYEO-XAH01-06	**Low**	Comment indicates functionality different from what is implemented
FYEO-XAH01-07	**Low**	Only the last symbol in memory is checked to be not 0
FYEO-XAH01-08	**Informational**	Hard to read functions impede maintainability
FYEO-XAH01-09	**Informational**	Inconsistent check of pair returned by getInnerTxn
FYEO-XAH01-10	**Informational**	Json value type is checked in assert before processing
FYEO-XAH01-11	**Informational**	Make sure that featureExpandedSignerList is enabled in Import::doSignerList()
FYEO-XAH01-12	**Informational**	The determineOperation function returns success for unknown operations
FYEO-XAH01-13	**Informational**	The function syntaxCheckXPOP does not verify that ledger::index exists
FYEO-XAH01-14	**Informational**	While loop that does not iterate

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

This is a code review of the xahaud release version for Xahau. Xahaud is forked from the original rippled implementation and this audit covers the new additions to the code since the last review of the hooks amendment by FYEO. This report does not cover a full audit of the entire rippled codebase, it covers only the code added to the codebase by XRPL Labs.

The audit was focused on analyzing the Xahau and hooks specific implementations. This included:

- The reward system
- The genesis hook and state
- Burn to mint
- Hooks installation
- Hooks execution
- The governance game hooks and their implementation.

Most of the identified issues are recommendations of best practices and small mistakes with low impact on security.

Several medium-risk vulnerabilities are related to:

- unsafe type casting
- low probability issues that cause undefined behavior

The high-risk vulnerabilities also cause undefined behavior but the probability of it is higher, which is why they are given a higher severity rating.

The code style is overall good and while it does use some c-style constructions these are generally in line with language specifications. Unit tests are extensive and check different scenarios for precompiled wasm hooks that are executed in a transaction testing environment.

The overall security of the project is good, no issues regarding possible assets loss or unauthorized access were found during this audit.

IMPORT TRANSACTION DOES NOT PROPERLY CHECK SIGNER LIST

Finding ID: FYEO-XAH01-01
Severity: **High**
Status: **Remediated**

Description

For the import transactions with multiple signers, the signers of the outer and inner transaction are not correctly compared.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 280

```
// check signer list
bool const outerHasSigners = tx.isFieldPresent(sfSigners);
bool const innerHasSigners = stpTrans->isFieldPresent(sfSigners);

if (!(outerHasSigners && innerHasSigners))
{
    auto const& outerSigners = tx.getFieldArray(sfSigners);
    auto const& innerSigners = stpTrans->getFieldArray(sfSigners);

    bool ok = outerSigners.size() == innerSigners.size();
    for (uint64_t i = 0; ok && i < outerSigners.size(); ++i)
    {
        if (outerSigners[i].getAccountID(sfAccount) !=
innerSigners[i].getAccountID(sfAccount) ||
            outerSigners[i].getFieldVL(sfSigningPubKey) !=
innerSigners[i].getFieldVL(sfSigningPubKey))
            ok = false;
    }

    if (!ok)
    {
        JLOG(ctx.j.warn())
            << "Import: outer and inner txns were (multi) signed with different keys.
"
            << tx.getTransactionID();
        return temMALFORMED;
    }

}
```

Severity and Impact Summary

This check does not compare the signers. The condition `!(outerHasSigners && innerHasSigners)` will evaluate to false if both are present.

Recommendation

Fix the if condition and return a failure in other cases.

INCONSISTENT CHECK OF ISFIELDPRESENT(SFTRANSACTIONRESULT)

Finding ID: FYEO-XAH01-02
Severity: **Medium**
Status: **Remediated**

Description

The `meta->isFieldPresent(sfTransactionResult)` is inconsistently implemented in. Sometimes it is directly accessed with no check, other places test for existence.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 52

```
if (!meta)
    return beast::zero;

auto const result = meta->getFieldU8(sfTransactionResult);
```

Accessed without check.

Line number: 205

```
if (!meta->isFieldPresent(sfTransactionResult))
{
    JLOG(ctx.j.warn())
        << "Import: inner txn lacked transaction result... "
        << tx.getTransactionID();
    return temMALFORMED;
}
```

Access with check.

Line number: 1275

```
if (meta->getFieldU8(sfTransactionResult) == tesSUCCESS)
{
    auto const tt = stpTrans->getTxnType();
    if (tt == ttSIGNER_LIST_SET)
        doSignerList(sle, *stpTrans);
    else if (tt == ttREGULAR_KEY_SET)
        doRegularKey(sle, *stpTrans);
}
```

Accessed without check.

Severity and Impact Summary

This variable should exist.

Recommendation

Implement checks to make sure it is set.

VALUES THAT WOULD HAVE CAUSED OVERFLOWS ARE IGNORED

Finding ID: FYEO-XAH01-03
Severity: **Medium**
Status: **Remediated**

Description

The code verifies there is no overflow before adding a number, but it is ignored if the value would have caused an overflow.

Proof of Issue

File name: src/ripple/app/tx/impl/SetHook.cpp
Line number: 551, 559, 562, 567

```
                if (paramBytes + entryBytes > paramBytes)
                    paramBytes += entryBytes;
//...
        if (hookFee + paramFee > hookFee)
            hookFee += paramFee;

        if (hookFee + createFee > hookFee)
            hookFee += createFee;
//...
    if (fee + hookFee > fee)
        fee += hookFee;

    return fee;
```

Severity and Impact Summary

Operations with high fee values may be accepted with a low fee.

Recommendation

Update the code to reject the hook if there is an overflow.

SYNTAXCHECKXPOP CHECKS VALIDATION UNL IN FOR LOOP MAKING FIELDS OPTIONAL

Finding ID: FYEO-XAH01-04
Severity: **Medium**
Status: **Remediated**

Description

In `syntaxCheckXPOP()`, there is a loop over `xpop["validation"]["unl"].getMemberNames()` which may have fields such as "public_key" but the test, as implemented, does not guarantee that these keys exists on `xpop["validation"]["unl"]`. Several statements in Import.cpp assume that they do.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 163

```
std::string strPk = (*xpop)[jss::validation][jss::unl][jss::public_key].asString();
...
auto const data =
base64_decode((*xpop)[jss::validation][jss::unl][jss::blob].asString());
...
auto const sig =
strUnHex((*xpop)[jss::validation][jss::unl][jss::signature].asString());
```

Also, `["validation"]["unl"]["manifest"]` is not checked to be base64, but assumed to be:

```
base64_decode((*xpop)[jss::validation][jss::unl][jss::manifest].asString())
```

Severity and Impact Summary

This could lead to the import of the transaction failing.

Recommendation

Make sure this data exists and is in the correct format.

XPOP_SLOT MAY WRITE DATA IN THE SAME SLOT

Finding ID: FYEO-XAH01-05
Severity: **Medium**
Status: **Remediated**

Description

The `xpop_slot` hook accepts slot id for `tx` and `meta`, but doesn't check if they are different.

Proof of Issue

File name: src/ripple/app/hook/impl/applyHook.cpp
Line number: 5580

```
DEFINE_HOOK_FUNCTION(
    int64_t,
    xpop_slot,
    uint32_t slot_into_tx, uint32_t slot_into_meta )
{
    HOOK_SETUP();

    if (applyCtx.tx.getFieldU16(sfTransactionType) != ttIMPORT)
        return PREREQUISITE_NOT_MET;

    if (slot_into_tx > hook_api::max_slots || slot_into_meta > hook_api::max_slots)
        return INVALID_ARGUMENT;
    ...
```

Severity and Impact Summary

In the case when the same (non-zero) slot id is passed, the `tx` value will be overwritten by the `meta` value.

Recommendation

Check that slots are different.

Comment indicates functionality different from what is implemented

Finding ID: FYEO-XAH01-06
Severity: **Low**
Status: **Remediated**

Description

This comment indicates functionality different from what is implemented.

Proof of Issue

File name: src/ripple/app/hook/impl/applyHook.cpp
Line number: 1215

```
// skip \0's
if (output[out_len-1] == '\0')
    out_len--;
```

Severity and Impact Summary

This will not skip any number of 0’s but just one.

Recommendation

Make sure this functionality is properly implemented.

ONLY THE LAST SYMBOL IN MEMORY IS CHECKED TO BE NOT 0

Finding ID: FYEO-XAH01-07
Severity: **Low**
Status: **Open**

Description

In `trace_float` the memory buffer is checked to not end with '\0'. In the case when multiple bytes are set to 0, the trace log may be cut.

Proof of Issue

File name: src/ripple/app/hook/impl/applyHook.cpp
Line number: 4463

```
    if (read_len > 0 && *((const char*)memory + read_ptr + read_len - 1) == '\0')
        read_len--;
```

Severity and Impact Summary

Possible loss of data in trace.

Recommendation

Check if all buffer is correct.

HARD TO READ FUNCTIONS IMPEDE MAINTAINABILITY

Finding ID: FYEO-XAH01-08
Severity: **Informational**
Status: **Open**

Description

There is some hard to read code, which hinders the maintainability of this codebase.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 468

```
if (!([](Json::Value const& proof, std::string hash) -> bool
{
    auto const proofContains =
    [](Json::Value const* proof, std::string hash, int depth = 0, auto proofContains =
nullptr) -> bool
    {
        if (depth > 32)
            return false;

        if (!proof->isObject() && !proof->isArray())
            return false;

        if (proof->isMember(jss::children))
            proof = &((*proof)[jss::children]);

        for (int x = 0; x < 16; ++x)
        {
            Json::Value const* entry =
                proof->isObject()
                    ? &((*proof)[std::string(1, "0123456789ABCDEF"[x])])
                    : &((*proof)[x]);

            if (entry->isNull())
                continue;

            if ((entry->isString() && entry->asString() == hash) ||
                (entry->isObject() && entry->isMember(jss::hash) &&
(*entry)[jss::hash] == hash) ||
                proofContains(entry, hash, depth + 1, proofContains))
                return true;
        }

        return false;
    };

    return proofContains(&proof, hash, 0, proofContains);
})((*xpop)[jss::transaction][jss::proof], strHex(computed_tx_hash_and_meta)))
{
    JLOG(ctx.j.warn())
```

```
        << "Import: xpop proof did not contain the specified txn hash "
        << strHex(computed_tx_hash_and_meta)
        << " submitted in Import TXN "
        << tx.getTransactionID();
    return temMALFORMED;
}

uint256 const computedTxRoot =
([](Json::Value const& proof) -> uint256
{
    auto hashProof =
    [](Json::Value const& proof, int depth = 0, auto const& hashProof = nullptr) ->
uint256
    {
        const uint256 nullhash;

        if (depth > 32)
            return nullhash;

        if (!proof.isObject() && !proof.isArray())
            return nullhash;

        sha512_half_hasher h;
        using beast::hash_append;
        hash_append(h, ripple::HashPrefix::innerNode);

        if (proof.isArray())
        {
            for (const auto& entry : proof)
            {
                if (entry.isString())
                {
                    uint256 hash;
                    if (hash.parseHex(entry.asString()))
                        hash_append(h, hash);
                }
                else
                    hash_append(h, hashProof(entry, depth + 1, hashProof));
            }
        }
        else if (proof.isObject())
        {

            for (int x = 0; x < 16; ++x)
            {
                // Duplicate / Sanity
                std::string const nibble (1, "0123456789ABCDEF"[x]);
                if (!proof[jss::children].isMember(nibble))
                    hash_append(h, nullhash);
                else if (proof[jss::children][nibble][jss::children].size() == 0u)
                {
                    uint256 hash;
                    if
(hash.parseHex(proof[jss::children][nibble][jss::hash].asString()))
                        hash_append(h, hash);
                }
                else
```

```
                        hash_append(h, hashProof(proof[jss::children][nibble], depth + 1,
hashProof));
               }
          }
          return static_cast<uint256>(h);
     };
     return hashProof(proof, 0, hashProof);
})((*xpop)[jss::transaction][jss::proof]);
```

Severity and Impact Summary

Hard to read code can impact the maintainability of the codebase.

Recommendation

Refactor the code to make it easier to understand.

Inconsistent check of pair returned by getInnerTxn

Finding ID: FYEO-XAH01-09
Severity: **Informational**
Status: **Remediated**

Description

For maintainability reasons, it is appropriate to check that neither of those values is null. There currently seems to be no case in which this would happen, but it may not be obvious to adjust this if the code in `getInnerTxn` is changed.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 183

```
auto const [stpTrans, meta] = getInnerTxn(tx, ctx.j, &(*xpop));

if (!stpTrans)
   return temMALFORMED;
```

There exists another caller which does implement it as such:

```
auto [tx, meta] = Import::getInnerTxn(applyCtx.tx, j);
if (!tx || !meta)
        return INVALID_TXN;
```

Severity and Impact Summary

A maintainability concern.

Recommendation

Implement checks in a consistent fashion.

JSON VALUE TYPE IS CHECKED IN ASSERT BEFORE PROCESSING

Finding ID: FYEO-XAH01-10
Severity: **Informational**
Status: **Open**

Description

`Json::Value` uses `assert` to validate if the field can be retrieved via `[]`.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 86

```
auto rawTx = strUnHex((*xpop)[jss::transaction][jss::blob].asString());
```

File name: src/ripple/json/impl/json_value.cpp
Line number: 849

```
const Value&
Value::operator[](const char* key) const
{
    JSON_ASSERT(type_ == nullValue || type_ == objectValue);
```

File name: src/ripple/json/impl/json_assert.h
Line number: 27

```
#define JSON_ASSERT(condition) \
    assert(condition);  // @todo <= change this into an exception throw
```

Severity and Impact Summary

Incorrect filed type may lead to `nullptr` dereference.

Recommendation

Consider throwing errors instead.

Make sure that featureExpandedSignerList is enabled in Import::doSignerList()

Finding ID: FYEO-XAH01-11
Severity: **Informational**
Status: **Remediated**

Description

It should be checked that `featureExpandedSignerList` is enabled before adding `sfWalletLocator` as a tag.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 991

```
        std::optional<uint256> tag;
        if (e.isFieldPresent(sfWalletLocator))
            tag = e.getFieldH256(sfWalletLocator);

        signers.emplace_back(
            e.getAccountID(sfAccount),
            e.getFieldU16(sfSignerWeight),
            tag);
```

File name: src/ripple/app/tx/impl/SetSignerList.cpp
Line number: 293

```
        if (signer.tag && !expandedSignerList)
        {
            JLOG(j.trace()) << "Malformed transaction: sfWalletLocator "
                               "specified in SignerEntry "
                            << "but featureExpandedSignerList is not enabled.";
            return temMALFORMED;
        }
```

Severity and Impact Summary

Adding this tag without the feature being enabled may lead to malformed data.

Recommendation

Add an explicit test to check the rules.

THE DETERMINEOPERATION FUNCTION RETURNS SUCCESS FOR UNKNOWN OPERATIONS

Finding ID: FYEO-XAH01-12
Severity: **Informational**
Status: **Open**

Description

The `determineOperation` function will return `tesSUCCESS` status even if it could not determine the operation.

Proof of Issue

File name: src/ripple/app/tx/impl/SignerEntries.cpp
Line number: 105

```
Operation op = unknown;
//...
if
//...
else if
//...
return std::make_tuple(tesSUCCESS, quorum, sign, op);
```

Severity and Impact Summary

This may lead to unexpected behaviour.

Recommendation

Consider addressing this if necessary.

THE FUNCTION SYNTAXCHECKXPOP DOES NOT VERIFY THAT LEDGER::INDEX EXISTS

Finding ID: FYEO-XAH01-13
Severity: **Informational**
Status: **Remediated**

Description

This statement `std::uint32_t(lgr[jss::index].asUInt())` directly accesses data which is not checked to exist in the `syntaxCheckXPOP` function.

Proof of Issue

File name: src/ripple/app/tx/impl/Import.cpp
Line number: 597

```
// compute ledger
uint256 computedLedgerHash =
sha512Half(
    HashPrefix::ledgerMaster,
    std::uint32_t(lgr[jss::index].asUInt()),
```

Severity and Impact Summary

This field should probably be checked to exist.

Recommendation

Check for the existence of this field.

WHILE LOOP THAT DOES NOT ITERATE

Finding ID: FYEO-XAH01-14
Severity: **Informational**
Status: **Remediated**

Description

This is a while loop that does not iterate.

Proof of Issue

File name: src/ripple/app/hook/impl/applyHook.cpp
Line number: 1160

```
while (read_len > 0)
{
    // skip \0 if present at the end
    if (*((const char*)memory + read_ptr + read_len - 1) == '\0')
        read_len--;

    if (read_len == 0)
        break;

    j.trace()
        << "HookTrace[" << HC_ACC() << "]: "
        << std::string_view((const char*)memory + read_ptr, read_len)
        << ": " << number;

    return 0;
}
```

Severity and Impact Summary

This code is not very maintainable as it is somewhat confusing.

Recommendation

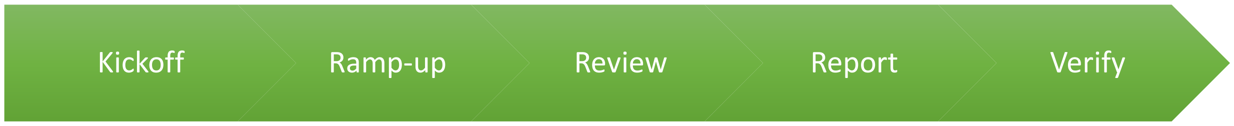
Implement a cleaner check.

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations