Threshold Multi-signature Audit

Cake Wallet and Thorchain

inference

20211220 – FINAL

Contents

1 Summary

Cake Wallet solicited Inference to review some changes introduced in Thorchain fork of the Monero cryptocurrency repository. Said changes are in the wallet RPC endpoint, allowing for more efficient threshold multi-signature generation. Inference engaged three persons familiar with threshold signatures to review these changes, in terms of cryptographic security and code safety.

This report presents the work performed, and notably describes:

- 2 potential security issues,
- 5 observations related to defense-in-depth, reliability, and performance.

The code review was done by threshold signature specialists Adrian Hamelink and Lucas Meier, with supervision by JP Aumasson.

We would like to thank Cake Wallet and Thorchain for trusting us.

2 Project overview

2.1 Scope

The client requested a review of the changes introduced by the thor_monero_signing_parallel branch in the thorchain/tss/monero-sign repository. The code was forked from github.com/monero-project/monero at commit c5c278c.

We reviewed the changes introduced between both of these versions, relating mostly to the following files.

- The RPC wallet server (`src/wallet/wallet_rpc_server`), which includes 4 new endpoints:
  - `check_transaction`
  - `sign_multisig_parallel`
  - `export_sigkeys`
  - `accumulate_multisig` [1]
- The `wallet2` component (`src/wallet/wallet2`), which includes 3 new functions:
  - `sign_multisig_tx_base`
  - `get_account_signing_pubkeys`
  - `acc_multisig_tx`

[1] The RPC function `on_check_transaction` is incomplete in the chosen branch, but was fixed in commit ae109c969. When reviewing this function, we took these modifications into account.

- The `ringct` component (`src`/`ringtc`/`rctSigs`) adds 1 public function `accMultisig`.

2.2 Goals

The Monero blockchain relies on *ring signatures*, whereby any member from a set of parties with public keys $\{K_1, \ldots, K_n\}$ can create a signature. The signer has a secret index $\pi \in \{1, \ldots, n\}$ and knows the private key k_π for the public key K_π. While the resulting signature does not reveal which party signed the message, the transaction should hide the amounts being transferred as well as the destination address. Moreover, they must also prevent double-spend attacks. Monero specifies several extensions to ring signatures which ensure some of these additional properties.

Most of Monero signature schemes admit a *multi-signature* variant, whereby the signing key is split amongst a group of N users. In this case, generating a secret for this key requires the participation of $M < N$ of the parties, for M a parameter called the *threshold*. The signature is generated by having each participant apply their secret share to a partial signature and pass along the resulting output to the parties who have not yet participated (note that when $M = N$, the signature can be performed in one round). This phase is completed once M participants have provided their input.

The main inefficiency of this approach is that the partial signature must be passed along to each signer before it is finalised, and can therefore cause a bottleneck due to the communication overhead. This is partly due to the fact that each signer must update their portion for each possible group of remaining signers to participate.

The changes we reviewed introduce an optimization to this threshold multi-signature flow that enables signers to perform their secret key operation independently, without waiting for the participation of previous parties. This requires each signer to know in advance which parties will participate, so that their contributions do not overlap. Once each signer has completed their share, the M partial signatures are broadcast throughout the group, and any party can combine them to generate the final signature for the group's signing key.

The entire patch affects 9 files, with modifications mostly to the wallet RPC server. This limits the potential vulnerability in the new code somewhat, since the changes must be compatible with existing specifications. For this reason, we paid particular attention to risks of private key extraction or leakage. We also assumed that the existing functionality, as described in the Monero documentation and existing code, is correct and secure.

3 Security issues

Severities are quantified with two dimensions, roughly defined as follows:

- Exploitability:
  - Low: Other vulnerabilities are needed to exploit the bug.
  - Medium: Privileged access is needed (for example, local), or costly attack (in terms of computation, storage, bandwidth, etc.)
  - High: Exploitation is easy (for example, remote, unauthenticated)
- Impact:
  - Low: The vulnerability exploit has clearly little to no impact on the system, its users, and operators.
  - Medium: Not low, but not high.
  - High: Critical system assets are clearly impacted, in terms of confidentiality, integrity, availability, or value.

Not that these definitions are vague on purpose, as they depend on the business context and assets at stake. For example, if service availability is critical to the system, then a DoS can qualify as high-impact.

3.1 S-CKW-001: [sign_multisig_parallel()] Unchecked parsing result

Exploitability: medium

Impact: low

3.1.1 Description

In wallet2.cpp#L7304, inside of the `sign_multisig_tx_base()` function, a public key is parsed from a string, but the return value of this parsing isn't checked:

```
1 crypto::public_key pubkey;
2 cryptonote::blobdata temp;
3 bool ret1=epee::string_tools::parse_hexstr_to_binbuff(el,temp);
4 bool ret=cryptonote::t_serializable_object_from_blob(pubkey,temp);
5 selected_keys.insert(pubkey);
```

because the default constructor for the `public_key` type doesn't initialize its content, a `char [32]`, the `pubkey` value is uninitialized. If parsing fails, the value may remain uninitialized, and then get used later. This is *undefined behavior* as per the C++ standard.

Exploitability: Triggering this code path is relatively simple: a malformed payload needs to be sent to the `on_sign_multisig_parallel()` RPC call. This can also be triggered by a node sending a malformed public key in a previous step of the signing protocol.

Impact: The impact of this is low, because there's no clear direct impact of this undefined behavior in that code. Furthermore, We could not identify a specific compiler version that would lead to an unsafe state. It is nonetheless prudent to eliminate this unpredictability factor.

3.1.2 Recommendation

The return value of these parsing functions should checked, and an error reported if they fail.

3.2 S-CKW-002: [accu_multisig()] Undefined Behavior with Empty Signatures

Exploitability: low

Impact: low

3.2.1 Description

In wallet2.cpp#L7427, and wallet_rpc_server.cpp#4361, the last signature is:

```
std::vector<rct::clsag> oldone = txshares.back().m_ptx.at(0).tx.
    rct_signatures.p.CLSAGs;
```

```
multisig_tx_set &tx_last = in_txs.back();
```

The code above accesses the last element of a list of pending transactions. This will cause undefined behavior if this list is empty, and that isn't checked yet in either case. In the second case, there is a check of the size right after, whose purpose is instead to verify that a threshold of signature shares has been provided:

```
multisig_tx_set &tx_last = in_txs.back();
THROW_WALLET_EXCEPTION_IF( in_txs.size()<m_multisig_threshold,
                           error::wallet_internal_error, "not enough
                               shares to accumulate");
```

Unfortunately, this check is done *after* a transaction is pulled from the list.

Exploitability: Directly calling the RPC endpoint with an empty set of pending transactions will trigger this path, but this may be difficult to achieve in a natural flow of the protocol, since nodes are responsible for gathering transactions themselves.

Impact: There's no clear direct impact of this vulnerability, although the undefined behavior resulting from accessing uninitialized memory is generally concerning.

3.2.2 Recommendation

In the second case, the size check can be moved above the first line, and in the first case, an additional "sanity" check that `txshares` is not empty should be added.

4 Observations

Here we list observations and suggestions not directly about security risks, but potential improvements, "defense-in-depth", quality assurance, and performance.

4.1 O-CKW-01: [Dockerfile] Leaked IP address

In the last line of the `Dockerfile` on the master branch, the arguments provided to the `monero-wallet-rpc` binary include an IP address. We recommend that this string be replaced by a benign IP, and removed from the git history.

4.2 O-CKW-02: [Dockerfile] unused ports

The updated Dockerfile only runs the `monero-wallet-rpc` client and relies on an external `monerod` daemon. Since the client only communicates over port `18083`, the remaining ports `18080` and `18081` can be removed from the configuration.

4.3 O-CKW-03: Unused variables

We found several points in the code where variables were left unused:

In rctSigs.cpp#L1642, the variable `all_shares` is unused.

In wallet_rpc_server.cpp#4361, the variable `oldone` is never used.

4.4 O-CKW-04: [RPC] `on_check_transaction()` can fail if two destinations have the same amount

In wallet_rpc_server.cpp#L4221, the `comp` function only compares the amounts of two `cryptonote::tx_destination_entry` elements, ignoring the other fields such as `addr` and `original`. If two destinations have the same amount but differ in some other other value, then the this RPC call will fail even though the destinations sets may be identical.

```
bool comp(cryptonote::tx_destination_entry &a,cryptonote::
    tx_destination_entry &b)
{
    return a.amount>b.amount;
}
```

An easy fix would be to check if the `amount` fields are equal, and if so return the lexicographical ordering on the `addr` fields.

4.5 O-CKW-05: [RPC] `on_check_transaction()` inconsistent failure handling

The function `on_check_transaction()` performs several checks and returns whenever it detects an inconsistency. There is a subtle difference between the RPC call failing or the transaction check failing. When the function returns **`false`**, this indicates that the RPC call failed and the response provided to the user will include an error code and message.

In some cases, such as the `validate_transfer()` check at wallet_rpc_server.cpp#4280, the RPC call is considered as failing, but no error message is provided to the caller. The two last checks which ensure that the destinations are the same will return **`true`** on failure, but will indicate this error in the response object.

We recommend that each check in `on_check_transaction()` behave similarly in order to minimize the possibility of a mishandled response for the client.

5 Disclaimer

This [security assessment] report (“Report”) by Inference AG (“Inference”) is solely intended for [Cake Wallet] (“Client”) with respect to the Report’s purpose as agreed by the Client. The Report may not be relied upon by any other party than the Client and may only be distributed to a third party or published with the Client’s consent. If the Report is published or distributed by the Client or Inference (with the Client’s approval) then it is for information purposes only and Inference does not accept or assume any responsibility or liability for any other purpose or to any other party.

Security assessments of a software or technology cannot uncover all existing vulnerabilities. Even an assessment in which no weaknesses are found is not a guarantee of a secure system. Generally, code assessments enable the discovery of vulnerabilities that were overlooked during development and show areas where additional security measures are necessary. Within the Client’s defined time frame and engagement, Inference has performed an assessment in order to discover as many vulnerabilities of the technology or software analyzed as possible. The focus of the Report’s security assessment was

limited to the general items and code parts defined by the Client. The assessment shall reduce risks for the Client but in no way claims any guarantee of security or functionality of the technology or software that Inference agreed to assess. As a result, the Report does not provide any warranty or guarantee regarding the defect-free or vulnerability-free nature of the technology or software analyzed.

In addition, the Report only addresses the issues of the system and software at the time the Report was produced. The Client should be aware that blockchain technology and cryptographic assets present a high level of ongoing risk. Given the fact that inherent limitations, errors or failures in any software development process and software product exist, it is possible that even major failures or malfunctions remain undetected by the Report. Inference did not assess the underlying third party infrastructure which adds further risks. Inference relied on the correct performance and execution of the included third party technology itself.