Security assessment

Alephium

inference

20220926 – FINAL

Contents

1 Summary

This report presents the results of our security assessment of Alephium's technology, covering the following core components:

- Cryptography primitives
- Wallet of the node
- Proof-of-less-work
- [De]serialization routines

We looked for security defects including:

- Software security bugs and unsafe issues
- Correctness of the code against specification and standards implemented
- Choice of cryptographic primitives and protocols
- "Supply-chain" risks
- Parameters type and size
- Randomness issues (generation, sampling, entropy, etc.)
- Deserialization abuse

Our methodology included manual code review and dynamic analysis based on modified versions of the test suite.

This report presents our findings, namely three low-severity issues, and an informational issue.

2 Detailed scope

We describe the different components covered by the audit, and their respective source code files.

2.1 Crypto

Cryptographic primitives in https://github.com/alephium/alephium/tree/master/crypto/src/main/scala/org/alephium/crypto, including AES, BLAKE2b, BLAKE3, Ed25519, BIP32 key derivation, and other algorithms. Most primitives rely on the BouncyCastle library (out of scope).

2.2 Serialization

Data serialisation/deserialization in https://github.com/alephium/alephium/tree/master/serde/src/main/scala/org/alephium/serde.

2.3 Proof-of-less-work (PoLW)

Mining operations in https://github.com/alephium/alephium/tree/master/protocol/src/main/scala/org/alephium/protocol/mining (esp. Emission.scala)

Difficulty adjustment in https://github.com/alephium/alephium/blob/master/flow/src/main/scala/org/alephium/flow/core/ (mainly ChainDifficultyAdjustment.scala)

This part involves matching the logic again that described in the paper.

2.4 Node wallet

Wallet code in https://github.com/alephium/alephium/tree/master/wallet/src/main/scala/org/alephium/wallet, with a focus on the secret's storage (in storage/SecretStorage.scala).

3 Security issues

3.1 S-ALE-01: [crypto/AES]: Random IV use risk

Exploitability: Low

Impact: Medium

3.1.1 Description

The IV is chosen randomly, which creates a risk of IV collision (and thus of plaintext leak) after approx. 2^{48} encryptions with the same key.

3.1.2 Recommendation

Reduce the IV size to 12 bytes.

Ensure that much fewer than 2^{48} encryption calls are done with the same key, otherwise pick the IV in a way that is not randomized, but ensure uniqueness (for example, a counter 0, 1, ..,).

3.2 S-ALE-02: [util/]: Timing leak in arithmetic functions

Exploitability: Low

Impact: Low

3.2.1 Description

Multiplication functions such as the following in U32.scala implement a “shortcut” if the first operand is zero (though not the second), as follows:

```
  def mul(that: U32): Option[U32] = {
    if (this.v == 0) {
      Some(U32.Zero)
    } else {
      val underlying = this.v * that.v
      if (U32.checkMul(this, that, underlying)) {
        Some(U32.unsafe(underlying))
      } else {
        None
      }
    }
  }
```

The execution time will thus significantly differ depending on whether the object is zero or not. This could leak information to an attacker if they can measure execution times of operations where the operand is a secret value.

3.2.2 Recommendation

Minimize the timing leaks by avoiding branchings.

3.3 S-ALE-03 [wallet] Default permissions for SecretFile

Exploitability: Low

Impact: Low

3.3.1 Description

The code in wallet/storage/SecretStorage.scala writes a (protected) `SecretFile` to the filesystem using default permissions, via `PrintWriter(file)`.

This may allow unauthorized parties to read and/or write to the file, thus corrupting its content.

3.3.2 Recommendation

Enforce `600` permissions.

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-ALE-01: [crypto/AES]: Unnecessarily long IV

AES.scala sets `private val ivByteLength = 64`, that is, a 64-byte IV for using with AES-GCM. However, a 12-byte IV is sufficient, and recommended in the official specs.

5 Disclaimer

This [security assessment] report (“Report”) by Inference AG (“Inference”) is solely intended for [insert name of client] (“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.