In response to “Quick Security Evaluation Perspectives”

Joop Ringelberg 19-10-22 Version: 1

Introduction

Radically Open Security has performed a basic quick security evaluation of the Alpha InPlace program produced by the Perspectives project. Their report, dated September 30th 2022, contains a number of recommendations. In this text I discuss them and outline the course of action I intend to take.

Devise a Formalized Threat Model

The author notes Perspectives lacks a formalized Threat Model. We take it that the current report, together with the text *A Security Perspective on the Distributed Runtime* (v4), forms the basis of an effort to compose that Threat Model.

Ensure Uniqueness of the GUID Identifying the User

When someone starts using InPlace, she should bring her own public-private key pair and make sure it is registered at some key server. Currently we intend to rely on PGP (Pretty Good Privacy) and PGP key servers.

If a message encrypted by a public key can *only* be decrypted with the corresponding private key, then a unique user identification may be obtained by encrypting a standard phrase (e.g. "user") with the public key the user brings to his InPlace installation.

Password Protect the user Account

Authenticating to InPlace is a purely local process (there is no server to authenticate to). It is similar to authenticating to one's laptop.

It should be noted that password protection is actually in place in situations where the users' data is stored in a Couchdb instance, mainly to cater for the case that this instance is on a remote server.

Only when the browsers' IndexedDB is used, is no password required. This is on the assumption that it is only the end user who has access to his machine and that this access is (password-)protected. Note that the user name involved is purely local; it will never leave the machine and neither does the password.

The remaining threat, then, is that a malicious agent acquires access to one's machine (either remotely or physically). Currently, we choose to communicate this risk with aspiring end users and not introduce passwords for IndexedDB data.

Securely Store Users' Passwords

This is a genuine concern and we will follow the recommendations.

Signing Keys Management

As the Perspectives system is distributed, nodes are equal and fulfil both client and server roles, if you wish to think in those terms. A transaction is signed on A (let's call that in a client role capacity) and verified on B (and that will then be in a server role capacity). Consequently, *every* node needs a way to look up the public key of *any* other node's user. We plan to outsource this administration of user id - public key association to existing key servers (though nodes may want to cache such associations). This opens up the issue of man-in-the-middle-attacks and we plan to address them as follows.

There are but two ways that Alice can become connected to Celia:

1. by sending her an Invitation file through a secure channel;
2. by being introduced to her, say through Bob.

Ad 1. After receiving an invitation, Celia should approach Alice (in person, or through a secure channel) and ask for her public key fingerprint. Celia can then verify that the public key she looks up on the key server is indeed owned by Alice. As Celia can now send encrypted messages to Alice, she can send her own fingerprint to Alice (and obviously Celia signs her Transactions, too). Alice can now verify the public key she looked up for Celia.

Ad 2. When Bob introduces Alice to Celia (and vice versa), he sends their fingerprints to either of them, both encrypted and signed. This is actually a built-in method for building a network of trust.

Encrypt Delta Messages

As public-private key pairs are used to sign Transactions, it would seem that these same keys can be used to encrypt transaction content (Delta). Assume Alice sends a transaction to Bob:

- Alice looks up Bob's public key on a third party key server;
- Alice encrypts the content of the transaction using Bob's public key;
- Alice signs the transaction with her private key;
- Bob looks up Alice's public key on a third party key server;
- Bob uses it to verify that the transaction did, indeed, come from Alice;

- Bob decrypts the contents of the transaction using his private key.

We will implement the above process.

Manage Third Party Dependencies (Supply Chain)

We have actually started using Dependabot, as recommended by ROS.

Window.postMessage considerations

We will implement the recommendations.