RPKI Testbed Requirements Document

Dec 8, 2009

Introduction

The RPKI testbed has two top level goals. 1. The testbed should provide the ability to automatically construct one or more “realistic” repositories according to the most current standards for RPKI certificates, CRLs, ROAs and manifests. It must take as input a configuration file that describes the statistical distribution of objects within each repository. 2. It must provide for the ability to act as a test repository for the RPKI software for both subsystem and system testing.

The purpose of the first goal is to be able to measure the performance of the RPKI software when invoked against a nominally correct (remote) repository or repositories such that the execution time for the RPKI software can be measured in a separable manner. More specifically, it should be possible to invoke the RPKI software against one or more of these testbed repositories and measure (a) the total network transfer time for rsync; (b) the total execution time of the synchronous components RPKI software, consisting of rsync_aur and rcli; and (c) the total execution time of the asynchronous components of the RPKI software, consisting of the garbage collector and chaser. The focus of the testbed for the first goal is on performance measurement; correctness of the repositories and also the RPKI software itself is assumed. Note that performance measurements should be able to be conducted for both absolute updates (one or more entire repositories is read in and processed) and also relative updates (only the changes from the last RPKI run are processed). Note that it is highly desirable to measure performance of the RPKI software on test repositories that closely resemble real world repositories in terms of their statistic properties. The degree to which this resemblance is achieved is the responsibility of the creators of the configuration file, as will be described in more detail below.

The purpose of the second goal is to be able to test the correct operation of the RPKI software under a variety of conditions, both nominal operating conditions and also error conditions. To date, functional testing of the RPKI software has primarily focused on unit tests. In unit testing, single points of failure are introduced and the software is run to insure that these failures are detected, and also that the actions of the software after detection do not permit errors to be introduced into the locally validated form of the repository (i.e., the database). Unit testing is necessary but not sufficient. The RPKI software must also be tested as a whole, which typically would involve both subsystem testing and also system testing. In subsystem testing, a particular part of the RPKI software functionality (e.g. manifests, or the router client/server interface) would be stressed by deliberately introducing edge cases, anomalies and errors into the subsystem under test. For example, a subsystem test for manifest processing might test conditions such as (a) remote repository does not have a manifest; (b) remote repository has a stale manifest; (c) remote repository has a manifest but also contains objects that are not on that manifest, and so forth. As such, subsystem testing is generally much more complicated than unit testing. By extension, system testing would encompass testing all subsystems at a single time, and could thus involve introducing several coordinated test conditions/errors/anomalies with the goal of determining if the software properly handles those conditions. In order to perform subsystem or system testing, the RPKI software must be run against one or more entire collections of objects, not just a single object as in unit testing. This produces the requirement that one can create such a collection (a remote repository) or collections automatically. This goal builds upon the first goal in that a typical test methodology would first produce one or more nominally correct repositories, verify proper operation of the RPKI software, and then manually perturb the repositories and observe the results. The first goal is thus focused on performance, while the second goal is focused on functionality. As an important corollary, the first goal is focused on verisimilitude to real-world repositories, while the second goal is focused on erroneous perturbations of those repositories that might or might not be expected to be encountered in the real world. (One would expect that subsystem and system testing would be prioritized to preferentially test those scenarios that might be expected to occur in the real world by, for example, inferring the types of operator errors that might be likely to occur and testing those first; in the ideal scenario, all cases would be tested to some level since all errors that can occur are likely to eventually occur somewhere.)

The remainder of this document will discuss the detailed requirements that must be met in order to achieve these two goals. Note that this is a requirements document only. It is not intended to be a design document, nor is it intended to provide cost estimates, although it is intended that those two activities be derivable from the contents of this document.

Performance Test Requirements

To meet the performance testing top level goal, the testbed creation software must be able to create a repository that is fully conformant with all requisite standards. It must also be possible to specify the statistical distribution of objects within the repository using a configuration file.

1. All certificates and CRLs in the testbed repository must conform to RFC 5280, and also the most recent SIDR document on resource certificates (currently, draft-ietf-sidr-res-certs-17.txt). In particular, all certificates must be valid X.509v3 certificates with address extensions conforming to RFC 3779. All SKIs in any created testbed repository must be unique. All URLs contained in any certificate extension (SIA, AIA, CRLDP) must be resolvable to valid locations. It must be possible to specify CRLDPs that are (a) within the directory hierarchy of the current publication point; (b) within the directory hierarchy of another valid publication point; (c) within a directory hierarchy that is outside any remote repository publication point. All URLs must be accessible to rsync.

2. All ROAs must conform to the most recent SIDR draft for the ROA format; at the present time this is draft-ietf-sidr-roa-format-06.txt.

3. All manifests must conform to the most recent SIDR draft for the manifest format; at the present time this is draft-ietf-sidr-rpki-manifests-06.txt

4. The testbed repository as a whole must conform to the most recent SIDR draft for the repository structure, currently draft-ietf-sidr-repos-struct-03.txt.

5. It must be possible to specify within the configuration file (a) the depth of the repository; (b) the width of the repository at each level; (c) the size of the repository, expressed as a total number of objects of each of the four types: (d) the distribution of address/AS# resources, expressed, for example, as a set of parameters for a named distribution function. It is expected that the information in the configuration file will have been prepared by a human cognizant of the real world statistical data associated with actual repositories and then represented in the particular format of the configuration file.

6. It must be possible to specify, via an out-of-band mechanism, trust anchor material. One possible method to implement this, for example, would be to provide the filenames and/or URLs of such material in the configuration file. It must be possible to represent the TA material in either the older format or the newer compound format (ETA+CMS).

7. The testbed creation software should log all its actions.

Subsystem Test Requirements

1. It must be possible to test the RPKI software against (a) a single repository on the local machine; (b) a single repository on a remote machine; (c) multiple repositories on any single machine; (d) multiple repositories on more than one machine.

2. It must be possible to introduce any existing certificate, CRL, ROA or manifest unit test case into a remote testbed repository and test the RPKI software against that repository.

3. It must be possible to test for arrival of certificates in a path in any order.

4. It must be possible to test for RFC 3779 failures; at a minimum, it must be possible to test the case in which a child certificate includes resources not included in its parent for any possible arrival order for the child and parent certificates.

5. It must be possible to test for the six “top level” manifest processing cases, which are: (a) nominal operation with a complete manifest; (b) no manifest; (c) stale manifest; (d) manifest containing objects not part of the publication point; (e) publication point containing objects not on the manifest; (f) publication point containing objects whose hashes do not match those on the manifest.

6. It should be possible to test all 24 manifest processing cases as defined in the RPKIv2 manifest testing document. At a minimum, the testbed creation software must not operate in a manner that would preclude any such testing.

7. It must be possible to test the “stale CRL” processing case (valid CRL not updated after its nextUpdate time passes).

8. It must be possible to test compound TA processing and also any unit test error cases devised for compound TA RPKI software, e.g. to incorporate said unit tests into the configuration for a testbed and observe the results.

9. It must be possible to test nominal and error cases for the RTR (router client/server) software, e.g. to incorporate any unit tests (nominal or error cases) for that protocol into the configuration for a testbed and observe the results.