Address Space PKI Test Document

Derrick Kong
4/26/07

Introduction

This document describes the set of testing to be performed on the deliverable software of the Address Space PKI (APKI) contract. Testing will be performed in three phases: unit (point) testing, performance testing, and system testing. The goal of unit testing is to test low level functionality of the APKI software in a comprehensive manner. Each unit test has been designed to test a single feature of the software in a manner orthogonal to other features. Performance testing, as the name implies, is designed to test the overall performance of the system under operational conditions, during a nominal interval of one day of elapsed time. All functions of the system should be able to be executed within this interval. System testing is designed to test the system as a whole, and consists of a series of functional tests that also correspond to normal actions that a user would perform on the system. The next three sections of this document describe the three types of tests in more detail.

Single Point Testing

For purposes of the ROA PKI, single point testing is defined to be verification of whether the system properly interprets and handles single input files. Legitimate files include route origination authorizations (ROAs), certificates, and certificate revocation lists (CRLs) which are syntactically correct and conform to the proper profile.

The single point test procedure will be conducted as follows:

A fully-compliant and syntactically correct ROA, certificate, and CRL will be selected from a set of sample data or manufactured from scratch. Each of these will then be modified in the following ways:

.

- A number of legitimate variations in content (as appropriate to the particular file) and will be created. These are intended to pass all verification tests to ensure that different legal options and variations are recognized.
- For each file type, one or more incorrect samples will be created to test the syntax parsing and content verification of each field within the file. These errors will include incorrect OIDs, illegitimate content, missing and duplicated fields, incorrect signatures, etc.
- For each file type, one or more samples will be created that will be syntactically correct (relative to the normative RFCs, in particular RFC3280), but do not adhere to the PKI profile. These are intended to be rejected by the system even if verified to be syntactically valid.

A complete list of all the file variations (both correct and incorrect) is available in Appendix A.

Once the sample set has been created, all the files will be input into the system for checking. The tester will be responsible for verifying that the system accepts all the correct variations and rejects all malformed and non-compliant versions.

Performance Testing

The ultimate goal of performance testing is to ensure that the users can effectively operate the APKI software and perform all necessary operations with it on at least a daily basis. This implies that it must be possible to perform all four principle operations – running rsync and rcli; running garbage collection; running the chaser client; and performing the comprehensive query on a reasonably sized database within a twenty-four hour period. The definition of a "reasonably sized database" is open for discussion, of course, but we are targeting a database that can store a few hundred thousand objects. In order to allow for expansion beyond this putative baseline, we would therefore like the set of four principal operations to fit comfortably within the twenty-four hour period.

Performance testing will test each of these four operations independently. We will not actually have a database with several hundred thousand real objects, but it is possible to create a database that simulates this size, as will be described below. Our target test size for performance and system testing is five hundred thousand objects.

First this document will describe how the database will populated, and then a description of each of the four tests will be given. The two basic ideas behind populating the test database are cloning and local database modification. Cloning is a very simple idea. If hello.cer is a valid certificate that has been entered into the database, then one can simply copy hello.cer to a different filename, say hello2.cer, and it can then be manually entered into the database. Since hello.cer was valid, hello2.cer will also be valid and since it has a different filename there will be no collision in the database itself. Of course hello2.cer will appear to be a child of hello.cer's parent (assuming hello.cer is not a trust anchor, of course). Cloning this gives us a way to add more-or-less identical records to the database. We can also modify the database records in place once we have added them using the interactive MySQL tool "mysql." To see how this works, consider the following example.

Suppose R1 is a ROA that is already in the database, with SKI(R1) = S1. There must then be an EE certificate E1 with SKI(E1) = E1, also in the database. All E1 antecedents (parents, grandparents, etc) must also be in the database. Let P be the parent of E1. If we now want to craft a ROA that appears to be different from R1 we do the following: copy R1 to R2 and E1 to E2. Enter E2 into the database; enter R2 into the database. In the database use the "mysql" tool to change the SKI of both the R2 entry and the E2 entry to some different value such that SKI(R2) = SKI(E2). P will still appear to be the parent of both E1 and E2, so we have partially modified the validation path for the ROA R2. Note carefully that we have not in any way modified the files R2 and E2. This artifice can obviously be done on any portion of the database. In fact, since “mysql” can be scripted, we can start with a small set of initial actually valid and unique entries and create an arbitrary number of database entries out of them. Since performance testing is only concerned with performance, this is deemed a completely reasonable approach.

The rsync performance test will consider six scenarios: populating an empty database with approximately 500,000 entries and adding 10% to an existing database, that is adding 50,000 entries to the (all numbers approximate). Each test will be timed using the Linux utility “time.” A fake repository will be set up as the source for the new objects. In order to simulate the delays associated with running rsync, this fake repository will be on a different machine than the test machine.

The garbage collection performance test will also rely on the ability to modify database entries, in this case the notAfter field in the certificate table and also the sn field in that same table. The “mysql” utility will be used to arrange that approximately 10% of the entries in the database will need to be expired or revoked. The garbage collection process will then be run on the command line and timed using “time”.

The query client performance test will invoke the query client in its “comprehensive query” mode over a database with approximately 10000 ROAs. The invocation will be timed using “time.” Cloning will be used to create the entries as in the previous two tests.

The chaser client performance test will be invoked over a database that has 500,000 entries, with the assumption that 1% of them have new SIA, AIA or CRLDP values that are not currently known to rsync. These new SIA, AIA and CRLDP entries will be created using database entry modifications as with the garbage collection test. The new URIs will point to fake repositories. “time” will be used to time each of the three tests.

System Testing

The goal of system testing for the APKI software is to verify that the top level functions of the system are being performed correctly. At the highest possible level, the system can be thought of as consisting of four components: the “rsync toolchain”, the “gc toolchain”, the “query toolchain” and the “chaser toolchain”. The rsync toolchain consists of rsync itself, the rsync_aur software, the rcli program, the underlying APKI database, and of course the DB infrastructure libraries that mediate access to the DB. The gc toolchain consists of the garbage collection client, the DB libraries and the DB; the query toolchain consists of the query client, the DB libraries and the DB; and, finally, the chaser toolchain consists of the chaser client, the DB libraries and the DB itself. Note that since the chaser toolchain can invoke the rsync toolchain, the chaser toolchain is, in some sense, a dependent of the rsync toolchain. Since all of the toolchains touch the DB libraries and the DB itself they are, of course, interdependent.

In the point tests, the individual functions of the system were tested. In the performance tests, the overall performance of each client was tested, as well as the (implicit) performance of the DB and its libraries. In system testing and end-to-end testing, tests will be performed that will simulate the operations of a typical user over the course of several days. The idea is to test for the presence of unexpected interactions between the toolchains, and also to test for the overall operational behavior of the system. In addition to attempting to duplicate the primary use case of the users, two areas will be singled out for more detailed examination: database table locking and resource utilization.

The nominal operational model for the four clients is that they are not expected to overlap in time. This cannot be guaranteed, however, so it could well happen that one or more clients are running at once. Each client is a user of the database so there is at least the theoretical possibility that such simultaneous accesses could create an issue. The stated behavior of MySQL is that it enforces locking on database accesses such that no collisions can occur, e.g. it is allegedly not possible to have two (or more) clients be incompatibly accessing a given set of entries at the same time. This will be verified by actually running the four clients on the simulated database (as described in the performance testing section) at the same time. Before this is done, however, the "mysqladmin" command line tool will be used to enable debugging in the MySQL APKI database. This causes debug information to be written for each transaction, including error information in case an error occurs. Once the simultaneous invocation of the four clients has completed, this debug log will be examined for the presence of any errors or warnings.

Several client programs, notably rcli, gc and chaser, are designed to run as daemon programs. This means that in normal operation they will be running perpetually in the background. Whenever a long-lived process is present on a system there is always the possibility that it will have a resource leak. The most common form of resource leak is a memory leak. If a program has a memory leak, and that program runs for a substantial period of time, it can eventually degrade overall system performance and even lead to system crashes. Other types of equally unfortunately resource leaks are also possible. A long-running program can leak file descriptors, for example, which will impact its functionality at the point when it can no longer open any new files. In short, any resource leak in a daemon program is undesirable. Fortunately there are a set of Unix/Linux utilities than can be used to track resources and explore potential resource leaks.

Each of the daemon processes will be left running for a period of several days. During this period statistics, using "ps", "vmstat", "iostat" and "lsof" will be collected on at least a daily basis. At the end of the weeklong test the resulting statistical data will be examined to determine if there are any pernicious trends, such as an every growing use of memory, an every growing number of open file descriptors, and so forth.

In addition to these two focused tests, general operational testing will also be done. The important aspect of this testing is to have observables. To this end, local file cloning and local database modification will be done so that the set of results to be expected is exactly known in advance. For example, in the fake repository all the new files to be added can be arranged to have filenames beginning with "add" as well as some deliberately invalid objects with filenames beginning with "bad". In the local database a set of scripted modifications will be done to ensure that a set of entries will expire today and that these entries (in the filename field in the database) will have a prefix "exp". Similar modifications will also be done to create a set of certificate table entries that will be revoked having the prefix "rev". Entries that chaser would notice can be crafted so that the resultant SIA, AIA, and CRLDP values all contain "new" within their URIs. Finally, ROA entries will be modified so that the resulting output of the comprehensive query will have a predictable pattern.

Once this staging of the test database has been done, a typical day in the life of the user will be performed. Each of the toolchains will be exercised in turn. After each tool is run, the contents of the database will be examined using the "mysql" command line tool, and the actual result compared with the expected result. Note will also be made of the time that each invocation takes. Finally when all four toolchains have been exercised (noting that chaser may reinvoke rsync, so that the rsync toolchain will be invoked at least twice) the overall expected result will be compared with the overall actual result. An operational timeline will also be developed based on the invocation times for each toolchain, and this timeline compared with the nominal one day timeline with gaps that is the hoped for outcome.

Appendix A: List of Single Point Failure Cases

```
General guide to bad certs:

Type 1: Missing Fields
  Null file
  Critical bits not set

Type 2: Extra/Duplicate Fields
  Various repeated fields, some the same, some different
  Critical bits set where they are not supposed to be

Type 3: Corrupted/bad Fields
  oids incorrect
  Bad signatures
  Bad IP addresses
  Bad AS numbers
  Invalid dates

Type 4: Out of Order Fields
  IP addresses not in order
  AS numbers not in order

Original good cert: FLXmgA2Ff9X7hUPpD9NIPosEoE
Additional EE cert: BghcBcD2mv4nKeK_dd9MnRWJKK4

 1.1: Zero length file

 1.2: First ID (issuer) missing
 1.3: Second ID (subject) missing
 1.4: Date missing
 1.5: PK missing
```

1.6: Basic Constraints field missing
1.7: CA boolean set/not set (for non-CA/CA)

1.8: Subject key ID field missing
1.9: Authority key ID field missing for not self-signed cert
1.10: Key usage field missing
1.11: keyCertSign/crlSign bits missing (for CA cert with no other bits set)
1.12: digitalSignature bit missing (for EE cert)
1.13: Certificate policies field missing
1.14: CRL Dist point field missing
1.15: Auth info access field missing
1.16: Subj info access field missing
1.17: IP Addresses and AS number fields BOTH missing

1.18: Basic Constraints crit flag missing
1.19: Key usage crit flag missing
1.20: Certificate policies crit flag missing
1.21: IP Addresses/AS numbers crit flag missing

2.1: First ID (issuer) duplicated
2.2: Second ID (subject) duplicated
2.3: Date duplicated
2.4: PK info duplicated
2.5: Basic Constraints field duplicated
2.6: Subject key ID field duplicated
2.7: Authority key ID field duplicated
2.8: Key usage field duplicated
2.9: Certificate policies field duplicated
2.10: CRL Dist point field duplicated
2.11: Auth info access field duplicated
2.12: Subj info access field duplicated
2.13: IP Addresses or AS number field duplicated

2.14: Subject key ID critical bit set
2.15: Auth key ID critical bit set
2.16: CRL dist point critical bit set
2.17: Auth info access critical bit set
2.18: Subj info access critical bit set

2.19: Auth key ID subfield (authorityCertIssuer) present
2.20: Auth key ID subfield (authorityCertSerialNumber) present
2.21: CRL dist point subfield (Reasons) present
2.22: CRL dist point subfield (CRLIssuer) present

3.1: 1.2.840.113549.1.1.11 (PK algorithm) changed
3.2: 2.5.4.3 (issuer) changed
3.3: End date same as start date (expired)
3.4: End date before start date
3.5: Date before 2049 encoded as Generalized Time (should be UTC) (RFC3280)
3.6: Date in 2050 or later encoded as UTC (should be generalized)
3.7: 2.5.4.3 (subject) changed
3.8: 1.2.840.113549.1.1.1 (PK alg not RSA) changed
3.9: PK changed (short/long)
3.10: 2.5.29.19 (basic constr) changed

```
 3.11: 2.5.29.14 (subj key ID) changed
 3.12: 2.5.29.35 (auth key ID) changed
 3.13: 2.5.29.15 (key usage) changed
 3.14: 2.5.29.32 (cert policies) changed
 3.15: 2.5.29.31 (CRL dist point) changed
 3.16: 1.3.6.1.5.5.7.1.1 (auth info access) changed
 3.17: 1.3.6.1.5.5.7.48.2 (auth info access data) changed
 3.18: 1.3.6.1.5.5.7.1.11 (subj info access) changed
 3.19: 1.3.6.1.5.5.7.48.5 (subj info access data) changed
 3.20: 1.3.6.1.5.5.7.1.7 (ip addr) changed
 3.21: 1.3.6.1.5.5.7.1.8 (AS num) changed

 3.22: Cert version not 3 (i.e. != 0x02)
 3.23: IP addresses with invalid values
 3.24: AS numbers with invalid values

 4.1: CRL dist has no rsync site
 4.2: CRL dist uses "RSYNC"
 4.3: Auth info access has no rsync site
 4.4: Auth info access uses "RSYNC"
 4.5: Subj info access has no rsync site
 4.6: Subj info access uses "RSYNC"

 4.7: IP addresses not in order
 4.8: AS numbers not in order

--------
ROA: mytest

 1.1: Empty file (pem header/footer only)
 1.2: Missing ContentType field
 1.3: Missing CMSVersion field
 1.4: Missing DigestAlgorithmIdentifiers field
 1.5: Missing EncapsulatedContentInfo field
 1.6: Missing SignerInfos field
 1.7: Missing eContentType field

 1.8: Missing SignerInfo version field
 1.9: Missing SignerIdentifier field
 1.10: Missing DigestAlgorithmIdentifier field
 1.11: Missing SignatureAlgorithmIdentifier field
 1.12: Missing SignatureValue field

 2.1: Duplicate ContentType field
 2.2: Duplicate CMSVersion field
 2.3: Duplicate DigestAlgorithmIdentifiers field
 2.4: Duplicate EncapsulatedContentInfo field
 2.5: Duplicate SignerInfos field
 2.6: Duplicate eContentType field
 2.7: Duplicate SignerInfo version field
 2.8: Duplicate SignerIdentifier field
 2.9: Duplicate DigestAlgorithmIdentifier field
 2.10: Duplicate SignatureAlgorithmIdentifier field
 2.11: Duplicate SignatureValue field

 3.1: Bad ContentType OID
 3.2: Bad CMSVersion number (not 3)
 3.3: Bad DigestAlgorithmIdentifiers (not SHA-256)
 3.4: Bad eContentType (not ROA)
 3.5: Bad eContentType info
 3.6: Bad SignerInfo version (not 3)
 3.7: Bad DigestAlgorithmIdentifier (not SHA-256)
 3.8: Bad SignatureAlgorithmIdentifier (not SHA-256 with RSA)
 3.9: Bad SignatureValue

[Not implemented:]
 4.1: Signed Data contents sequence out of order (24 possible variants)
 4.2: eContent contents out of order:
  4.2.1: asID after ipAddrBlocks
  4.2.2: addressFamily after addressesOrRanges
 4.3: signerInfo sequence contents out of order (120 possible variants)


-----------------------------------------------------------------------

CRL: s0Rk925AmjX-pu8WWN9FOXuHz8Q

 1.1: Blank file
 1.2: Missing issuer name
 1.3: Missing issue/next issue date
 1.4: Missing AKI
 1.5: Missing CRL number
 1.6: Missing PK

 2.1: Duplicate issuer
 2.2: Duplicate date
 2.3: Duplicate AKI
 2.4: Duplicate CRL number
 2.5: Duplicate PK

 2.6 CRL number field with crit bit set

 3.1: Version number not 2
 3.2: Cert OID corrupt
 3.3: Issuer OID corrupt
 3.4: AKI OID corrupt
 3.5: AKI data corrupt
 3.6: CRL number OID corrupt
 3.7: CRL number data > 20 bytes
 3.8: PK OID corrupt
 3.9: PK data corrupt

 3.10: Next issue date same as issue date
 3.11: End date before start date
 3.12: Date before 2049 encoded as Generalized Time (should be UTC)
(RFC3280)
 3.13: Date in 2050 or later encoded as UTC (should be generalized)
```