Existing Trust Storage Implementations

Obviously if a comprehensive, future-proof and realistic standard representation of out-of-band trust policy exists, we should not define a new representation for Linux. Instead we should gather around it. So let's examine the various representations in use, and why they are insufficient to provide such a comprehensive standard.

1. NSS Trust Objects

Internally NSS represents out-of-band trust policy using PKCS#11 trust objects. These are not well documented [5] so an attempt will be made to describe them here.

Each NSS trust object contains the following attributes [6] used to find the the trust object that applies to a given X.509 certificate:

`CKA_CLASS`

> `CKO_NSS_TRUST`

`CKA_CERT_SHA1_HASH`

> A SHA1 hash of the DER encoded X.509 certificate to which this trust object's policy applies.

`CKA_CERT_MD5_HASH`

> An MD5 hash of the DER encoded X.509 certificate to which this trust object's policy applies.

`CKA_ISSUER`

The DER encoding of the issuer of the X.509 certificate to which trust object's policy applies.

`CKA_SUBJECT`

The DER encoding of the subject of the X.509 certificate to which trust object's policy applies.

`CKA_SERIAL_NUMBER`

The DER encoding of the serial number of the X.509 certificate to which trust object's policy applies.

The NSS trust object then contains the following usage attributes. Together these roughly represent the KeyUsage and ExtendedKeyUsage certificate extensions, as out-of-band trust policy. The names should be self explanatory for readers familiar with those certifiacte extensions.

- `CKA_TRUST_DIGITAL_SIGNATURE`
- `CKA_TRUST_NON_REPUDIATION`
- `CKA_TRUST_KEY_ENCIPHERMENT`
- `CKA_TRUST_DATA_ENCIPHERMENT`
- `CKA_TRUST_KEY_AGREEMENT`
- `CKA_TRUST_KEY_CERT_SIGN`
- `CKA_TRUST_CRL_SIGN`
- `CKA_TRUST_SERVER_AUTH`
- `CKA_TRUST_CLIENT_AUTH`
- `CKA_TRUST_CODE_SIGNING`
- `CKA_TRUST_EMAIL_PROTECTION`

The above usage attributes each can contain a trust setting, one of the following:

`CKT_NSS_TRUSTED`

The certificate is trusted for this usage.

`CKT_NSS_TRUSTED_DELEGATOR`

The certificate is trusted anchor as a certificate authority for the usages.

In NSS the trusted anchor does not have to be self-signed. You can explicitly trust an intermediate certificate as if it were a trusted anchor.

`CKT_NSS_NOT_TRUSTED`

This certificate is explicitly not trusted: neither for the given usage, nor as a delegator of the given usage.

`CKT_NSS_MUST_VERIFY_TRUST`

The certificate is not a trusted anchor (even if a later trust record in another PKCS #11 module says this cert should be trusted). If the marked certificate is self-signed, then this is semantically equivalent to `CKT_NSS_NOT_TRUSTED`, except NSS will return a different error code (*unknown CA* for `CKT_NSS_MUST_VERIFY_TRUST`versus *untrusted CA* for `CKT_NSS_NOT_TRUSTED`).

`CKT_NSS_TRUST_UNKNOWN`

This record does not explictily provide trust one way or the other. If there is another trust record for this cert in another PKCS #11 module, use it.

1.1. Deficiencies

NSS trust objects have been around for nearly two decades. They may have been sufficient in the past but are showing their age.

These trust objects do not seem to be designed as a comprehensive representation of out-of-band trust policy. They are insufficient in the following ways:

- Makes use SHA1 and MD5 hashes both of which are aging cryptographically.
- Trust objects only support trust policy related to the KeyUsage, ExtendedKeyUsage and parts of the BasicConstraints certificate extensions.
- Even though the ExtendedKeyUsage certificate extension can support arbitrary usages, the set of usages represented by these trust objects is limited to those defined above. Trust policy for additional usages is awkward to add.
- Blacklisting is done by marking a certificate as untrusted for specific usages. This works in practice but does not correctly model the reality of having a certificate blacklisted completely and for any usage.
- Trust objects are a PKCS#11 specific. While PKCS#11 is one acceptable object model for representing out-of-band trust policy, for a standard representation it cannot be the only one.

2. OpenSSL Trusted Certificates

OpenSSL contains a representation of out-of-band trust policy in its`TRUSTED CERTIFICATE` PEM blocks aka. CertAux. Files containing this information can be manipulated using its **openssl x509** tool.

It appears that this format is undocumented, so an attempt will be made to document it here.

PEM files contain a header and footer containing the words `TRUSTED CERTIFICATE`. Contained in the PEM data are two DER sequences. The first is an X.509 certificate, and the latter is a structure known internally as `X509_CERT_AUX`.

The `X509_CERT_AUX` DER sequence may be defined as follows:

```
CertAux ::= SEQUENCE {
      trust      SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
      reject     [0] SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,
      alias      UTF8String OPTIONAL,
      keyid      OCTET STRING OPTIONAL,
      other      [1] SEQUENCE OF AlgorithmIdentifier OPTIONAL
}
```

The `trust` and `reject` fields contain sequences of ExtendedKeyUsage object identifiers to trust the certificate to be used for, or to reject usage of the certificate for.

Together `trust` and `reject` fields represent out-of-band trust policy representing the ExtendedKeyUsage certificate extension. The other fields are not related to trust policy.

2.1. Deficiencies

This representation seems to be designed to solve a specific use case, and not designed as a comprehensive way to represent out-of-band trust policy. It is insufficient in the following ways:

- This format only supports trust policy related to the ExtendedKeyUsage certificate extension.
- Blacklisting is done by rejecting a certificate for specific usages. This works in practice but does not correctly model the reality of having a certificate blacklisted completely and for any usage.
- This format has OpenSSL implementation specific traits. The PEM contents are the concatenation of two DER structures, and though trivially parseable with the OpenSSL DER parser, it is awkward to parse especially when using other and/or strict DER parsers.

3. Trust Assertions

Trust Assertions are the author's previous attempt to solve the problem of sharing trust policy information. Details about this are available online [7].

3.1. Deficiencies

Although claiming to solve the problem of out-of-band trust policy in a general way, closer inspection and application to the real world exposed the following problems:

- This concept only supports trust policy related to the ExtendedKeyUsage certificate extension.
- Blacklisting is done by rejecting a certificate for specific usages. This works in practice but does not correctly model the reality of having a certificate blacklisted completely and for any usage.
- Although they claim to be general trust assertions were thought out as a PKCS#11 specific concept. While PKCS#11 is one acceptable object model for representing out-of-band trust policy, for a standard representation it cannot be the only one.

In addition claims of extensibility and generality proved hard to implement in the real world, and trust assertions ended up as a far more constrained concept that initially envisioned.

4. Certificate Authority Bundles

A bundle is either a file or directory containing one or more X.509 certificate authorities. These have been used to represent the possible anchors on a system. These are widely used today.

They are usually stored in the OpenSSL PEM format, but may also be seen in the Java Keystore format, and others.

4.1. Deficiencies

Although widely used today certificate authority bundles have the following deficiencies as a standard representation of trust policy:

- There is no standard way to represent out-of-band trust policy in addition to the policy contained in the certificate extensions. In theory one could create different bundles for certificate authorities

trusted for different usages and circumstances, but this quickly gets out of hand.

- There is no concept of blacklisting in a such a bundle bundle. One can remove a certificate from the bundle, but if that certificate is used in the middle of a certificate chain rather than as an anchor, the certificate validation will not respect such a removal.

---

[5] Although one can see them in the NSS source code certdata.txt file in all their glory.

[6] In addition to standard PKCS#11 object attributes

[7] Storing Trust Assertions in PKCS#11 Modules