Discussion Summary 2016-08-01

Discussion Summary 2016-08-01

Subject: avoiding dynamic metadata queries

See the archives for the complete discussion thread: avoiding dynamic metadata queries

1. The Metadata Query Protocol is a scheme for addressing metadata resources, it is not a metadata query language per se.
2. Everything of interest is known in advance and can therefore be pre-computed. In particular, signed per-entity metadata can be pre-computed.
3. The current MDQ protocol spec mandates that an MDQ server MUST serve up an "all entities" aggregate at a pre-defined endpoint. At the protocol level, this requirement should be relaxed: An "all entities" aggregate MUST be served at such-and-such an endpoint location but a server is NOT REQUIRED to provide such an aggregate.
4. Our initial deployment of an MDQ server should not expose an "all entities" aggregate. We want deployers to migrate to the MDQ server to leverage per-entity metadata, not a full aggregate. Those deployments that need or want a full aggregate should continue to get it from md.incommon.org, at least for the foreseeable future.

Subject: implementing a cache on the client

See the archives for the complete discussion thread: implementing a cache on the client

MDQ Client Software

Possible additions to the MDQ Client Software wiki page:

1. There are things we could do on the client side to increase the effectiveness and reliability of MDQ in practice:
    a. Implement on-disk caching, that is, “backing files” of per-entity metadata that quickly and safely initialize the in-memory cache and avoid “dead in the water” scenarios at system startup
    b. Implement interval-based refresh, as opposed to simply expiring entries and having user interaction trigger a refresh
2. One possible strategy is to avoid client-specific optimizations (in particular, Shibboleth-only optimizations) in favor of broad client support for the MDQ protocol
    a. A client software feature or optimization should be implemented if and only if client implementations are required to support it. (This suggests that we revisit our implementation profile with respect to MDQ.)
    b. If a goal is broad client support for the MDQ protocol, then that implies that we MUST deploy TLS on the MDQ server since some clients (such as AD FS) will not fetch SAML metadata at a location that begins with “http://”.
3. An alternative strategy is to extend client software to help bridge the gap during the transition to per-entity metadata. This will help build and maintain trust as the MDQ server is deployed and hardened.
4. Perhaps the best overall strategy is to focus effort on the server-side deployment (not client optimizations)
5. What is our response to these FAQs: What is Plan B if the MDQ server fails? Should I (as a deployer) load the aggregate up front?
    a. If a significant number of deployers end up loading the aggregate up front, whether that be for discovery or reliability, then our overall deployment of MDQ (both client and server) will be deemed a failure.
6. Pushing metadata out to the endpoints is a last resort option (with its own set of issues and risks)

MDQ Server Software

Possible additions to the MDQ Server Software wiki page:

1. TLS on the MDQ server can not be avoided if we truly want to be all-encompassing on the client side. AD FS, for example, will not consume SAML metadata at an endpoint location that begins with “http://”.
2. A fundamental decision with respect to TLS on the MDQ server: Do no harm or actively prevent harm? By hosting over http only, you force the issue, at the cost of guaranteeing no security at all for a wide range of software. It's a trade off with no clear answer.
3. If we want to embrace MDQ clients other than Shibboleth and SSP in our deployment plan, then we must configure TLS on the MDQ server. If we're going to do that, we should do it in such a way as to provide actual security benefit to the consumer.
4. One possible approach: Create a TLS key and certificate signed by the key used to sign per-entity metadata. Protect the MDQ server with this TLS key and certificate. Then clients can choose between two equivalent trust models, XML Signature or TLS. In each case, the client trusts the same public key certificate.