A [D]TLS-based GSS Mechanism

REQUIRED Goals

- MUST provide GSS-API semantics
- MUST adhere to PKIX
- Must be reviewed

Desirable Goals

- Channel binding support
- Easy to *review*
- Easy to *implement*
  - Including ***kernel-mode*** implementation of per-message tokens. After all, the NFSv4 community wants this mech, and several implementations put per-message tokens in OS kernel-land.
- Support for PKIX-specific name types
- Support for existing use of existing certs

Quick Sketch: Sec Context Tokens

- Use TLS **as is**
  - Don't “decorate on the outside”
  - Except for the standard header on initial context tokens
- TLS handshake protocol messages → GSS mech context tokens
  - ClientHello → initial context token
- Use TLS extensions for channel binding, asserting names, indicating acceptor name
  - RFC4680-based extensions

Benefits of Using TLS

a) Much simpler to specify than SPKM-type designs

b) Much simpler to review and analyze also

- Assume that TLS is OK, go from there

c) Specification re-use → implementation re-use

- There exist plenty of TLS off-the-shelf implementations

d) TLS exts. will benefit non-GSS TLS apps too

Quick Sketch: Channel Binding

- TLS ext., like RFC4681, based on RFC4680
- Client and server tell each other that they want to do channel binding in their `Hello`s
- Channel bindings sent in `SupplementalData` extension (see RFC4680)
    - Or not sent, as long as they're included in the Finished message computation!
- GSS semantics, even krb5 mech semantics
- **OPTIONAL**

Quick Sketch: Naming

See also naming presentation

- [OPTIONAL] TLS ext. for asserting a `GeneralName`
  - Or, rather, *index* of name. See naming preso.
  - `SupplementalData` (see RFC4680)
- [OPTIONAL] TLS ext. for indicating the desired target name
  - Like TLS ServerName indication, but more general
- Exported name token format, default name selection → see naming presentation

Quick Sketch: Per-msg Tokens

- TLS record protocol messages don't provide out-of-sequence processing support needed for GSS-API
  - DTLS does
- We can either
  - Use DTLS record protocol for per-msg tokens
  - Re-use RFC4121 (krb5 mech) per-msg tokens
    - Or krb5 for some cipher suites and DTLS for the rest

Quick Sketch: Per-msg Tokens

- Using DTLS record protocol messages for per-msg tokens → pure TLS-based mech
- But re-using krb5 mech per-msg tokens would greatly simplify implementation for NFSv4
  - Since NFSv4 implementations tend to be kernel-mode and they tend to implement GSS per-msg token processing in kernel-land
    - Linux, *BSD, Solaris, ONTAP
    - Same may apply to CIFS

On Per-msg tokens

- DTLS pros
  - Gets us new TLS cipher suite additions for free
- DTLS cons
  - Less available than TLS?
  - How many off-the-shelf kernel-land record protocol implementations?

On Per-msg tokens

- Re-using Kerberos V – pros
  - Readily available implementations, including kernel-land implementations
  - Gets us new Kerberos V enctype additions for free
- Cons
  - Not pure TLS...
  - Is TLS likely to get new ciphersuites faster than Kerberos V is likely to get new enctypes? Probably
    - So what?

On Per-msg tokens

- Or do both! And negotiate which one through a TLS extension.
  - One can be REQUIRED by the spec, the other can be OPTIONAL
  - Or maybe REQUIRE use of the Kerberos V mech's per-msg tokens for when the negotiated TLS cipher suite has a close-enough equivalent Kerberos V enctype **today**
    - e.g., AES w/ HMAC-SHA-1

GSS-TLS Sketch: Putting it all together

- TLS handshake messages → context tokens
  - Prepend standard GSS initial context token header to ClientHello
- Channel binding as a TLS extension
- TLS extensions for asserting peers' intended canonical name and for initiator to indicate intended acceptor name
- TBD: Per-msg tokens: krb5 per-msg tokens vs. DTLS record messages

Misc Details, Q/A

- Need GSS QoPs for TLS cipher suites
  - Need GSS extensions to make QoPs usable though
- Obviously, a TLS-based mech would support `GSS_C_NT_ANONYMOUS`