New GS2 Design

- Lighter weight design achieved by:
  - Dropping sec layers (no nego of layers and maxbuf)
  - Replacing original GS2 channel binding (CB) semantics
    - Before: CB success/failure affected sec layer negotiation
    - Now: CB is still negotiable, but all-or-nothing when used
- Pretty mechnames (not derived from OIDs)
  - unless the mech didn't have a pretty mechname from the get-go

New GS2: Headers

- GS2 now consists of two simple headers:
  - one prefixed to the first client→server message
  - one prefixed to the application's CB data
- 1st client message header:
  - A one-byte (or bit) constant
  - A one byte (or two bit) flag for CB
  - SASL authzid
- CB header:
  - Same as 1st client message header!

New GS2: Headers

- The constant flag is for compression of the GSS-API initial context token pseudo-ASN.1/DER mech OID header (see RFC2743, section 3.1)
- The channel binding flag is for CB negotiation and downgrade detection (see later slides)
- The authzid is needed because GSS-API mechs don't have an equivalent
  - (All other SCRAM message components other than authzid and CB flags stay in SCRAM)

New GS2 Design: Headers

- The RFC2743 header compression flag is an ASCII 'T' or 'F' and is always 'T' for SCRAM
  - 'T' → mech is “standard” mech per-RFC2743
- The CB flag is an ASCII 'n' (client can't), 'y' (client could but didn't) or 'p' (“present”, i.e., channel binding was used)
- The authzid is: “a=” saslname “,”
- Trivial ABNF

New GS2: Mechnames

- SCRAM will have two mech names:
  - SCRAM-SHA-1-PLUS
  - SCRAM-SHA-1
- The “-PLUS” suffix comes from GS2 and is for downgrade detection (see next slide)
- Pretty mechnames; GSS_SASL_mechname() function added for looking up a GSS-API mech's SASL mechname
  - New mechs should specify/register a pretty SASL mechname or they will get a OID-derived name

New GS2: Downgrade detection

- CB needs to be negotiable
  - Use two mechnames (see previous slide)
- Need to securely deal with: client app & SASL/ TLS stack supports CB but server doesn't
- Remember: SCRAM has no sec layers
  - listing the server's SASL mechs after authentication cannot protect the negotiation
-

New GS2: Downgrade detection

- Server advertises SCRAM-SHA-1 or both, SCRAM-SHA-1 and SCRAM-SHA-1-PLUS
  - Only SCRAM-SHA-1 if the server can't do CB
  - Both if it can
- Client picks one and sets the GS2 CB flag:
  - 'n' if client can't do CB, 'y' if it could but the server only advertised SCRAM-SHA-1, 'p' if the client used CB regardless of what was advertised
    - If client sees only SCRAM-SHA-1 it will not do CB

New GS2: Downgrade detection

- If client uses CB and the server can't: authentication fails, obviously
- If client couldn't use CB: authentication succeeds IFF the server couldn't either
- If client could have used CB but didn't, and the server did support CB: authentication fails because of downgrade attack

New GS2: Authenticated plaintext

- GS2 adds a header to the client's first message. This needs to be authenticated.
- GS2 header is authenticated by always using the GSS-API channel binding facility, with any actual CB data prefixed with the GS2 header.
  - Only GSS-API mechanisms that support channel binding need apply
  - krb5 does, SCRAM does, PKU2U will (but PKU2U is not needed – TLS with user certs will suffice)

New GS2: Analysis

- The RFC2743 header compression flag is constant in SCRAM case
- CB negotiation is needed no matter what
  - A pure SASL SCRAM could do CB nego in the mech instead of via mech nego
- The authzid is needed whether GS2 is used or not (GS2 “lifts” authzid out of pure SASL SCRAM)
- Conclusion: GS2 is now as simple as it gets

New GS2: SASL API impact

- SASL APIs will need:
    - “Can do CB” input
    - Actual CB data input
- This would be the case with or without GS2
- Also useful: “server SASL mech list” input on client side