On Channel Bindings

draft-ietf-nfsv4-channel-bindings-02.txt

Nicolas.Williams@sun.com

(to be presented at 60th IETF NFSv4 WG and KITTEN BoF meetings)

Introduction

- Channel bindings allow session protection at one network layer to be delegated to session protection at another by proving that there is no MITM at the lower layer
- Why? Performance *plus* security.
- Concept first described in GSS-APIv2 (see rfc2743 and rfc2744)
  - But specs were lacking

Formal Definition (rough; see I-D)

- Mutual authentication at app-layer
- App-level end-points exchange integrity-protected proof of knowledge of “channel bindings” for lower layer, **secure** channel
- Channel bindings data “name” a channel
  - ***must*** be cryptographically bound to the named channel

Examples: TLS, SSHv2

- Channel bindings for TLS: client and server finished messages
- Channel bindings for SSHv2: session ID
- These are cryptographically bound to the initial TLS or SSHv2 key exchange
  - SSHv2 re-keys are bound to the initial key exchange
- {TCP, SCTP, UDP}/IPsec? It can be done – see later slides
- NULL bindings? Better than AUTH_SYS...

The GSS-API & Channel Bindings

- RFC2743 speaks of channel bindings
  - Provides no structure, just “OCTET STRING” and little guidance
- RFC2744 provides C (!) structure
  - And little guidance beyond bindings to network addresses
- GSS-API channel bindings are not negotiable
  - Either apps use them, or don't

The GSS-API & Channel Bindings (cont.)

- To make GSS channel bindings useful we
  - Provide a generic structure for channel bindings data based on rfc2744's C struct and rfc1964's language-neutral interpretation of same
  - Provide guidance, specs[*] for several types of channel bindings (to TLS, SSHv2, Ipsec)
  - Provide for negotiation of channel bindings by adding new stackable GSS pseudo-mechs and using same to leverage existing negotiation of GSS mechs
    - Apps offer/select these mechs when they have bindings

Benefits: Overview

- Avoid double encryption when possible, e.g.,
  - SSHv2 over IPsec
  - SASL over TLS
  - NFS over IPsec, SSHv2, etc...
  - Leverage IPsec acceleration in HW
  - Remember: secure binding of two channels
- Reduce number of active crypto contexts (NFS)
- Facilitate RDDP over IPsec

And w/o Channel Bindings?

- If the lower layer's authentication facilities satisfy applications needs then there's no need for channel bindings
- But we expect IPsec w/ user certs to be rare
  - And GSS-API extensions to IKEv2 to be slow in coming to market
- Plus, apps which multiplex multiple users onto one connection, as NFS does, can't use IKE authentication
  - And one conn. Per-user, for NFS, is a non-starter

Performance Benefits: NFS

- NFS clients typically establish more GSS-API security contexts than they absolutely must
  - Several per-{user, client, server}; adds up!
- With channel bindings none of those contexts are used for session protection
  - Fewer active crypto contexts → typically lower crypto HW overhead
- Leverage HW-acceleration at lower layers (IPsec)

Performance Benefits: RDDP

- RDDP layers between the transport and the application to facilitate receiver zero-copy by addressing interesting buffers in app payloads and directing RNIC to directly place data
- App data must be in cleartext relative to RDDP header, else app-layer crypto must be supported by RNICs (no way)
  - Channel bindings makes this possible
  - Some RNICs can be expected to accelerate ESP/AH

Performance Benefits: NFS w/ RDDP

- Duh!

What about IPsec?

- What's an IPsec channel?
  - A TCP (or SCTP) connection protected with transport-mode SAs with same protection/ authenticated IDs for duration of connection
  - A UDP datagram protected by transport mode SA
  - etc...
- Apps need new APIs to deal with IPsec channels

What about IPsec? (cont.)

- Channel Bindings data for IPsec:
  - SA IDs authenticated by key exchange protocol
    - ***Latched*** in SPD for connections to the connections' traffic selectors (i.e., protocol #, port #s)
  - Protection parameters
    - ESP or AH, enc algorithms
  - Traffic selectors for connection/datagram
    - protocol number, port numbers (SCTP has more)
- Cryptographic binding is indirect, through authentication, APIs, SPD

What about IPsec? (cont.)

- Apps need APIs to retrieve/specify some of these items, see:
  - draft-ietf-ipsp-ipsec-apireq-00.txt
  - draft-ietf-nfsv4-channel-bindings.txt

What about Anonymous IPsec?

- Huh? Anonymous IPsec? An oxymoron?
  - No! Apps that provide for authentication may not care about IDs authenticated by IPsec.
    - And why should one have to deploy multiple authentication infrastructures?
- With IPsec IDs as part of the bindings anon IPsec can be constructed thusly
  - With ***non***-pre-shared, self-signed certs
  - Use cert public keys as IDs
  - Policy should allow apps like NFS to use this

Channel Bindings Structure, Constructor Functions

- draft-williams-gssapi-channel-bindings-00.txt
  - Not yet published; missed cut-off for this meeting
- Generalizes rfc2744 C structure of bindings
- Specifies bits to be passed to GSS-API for channel bindings for TLS, SSHv2, IPsec
- Specifies utility contructor function APIs for formatting same

CCM-BIND

- GSS *pseudo*-mechanism
  - *Stacks* atop concrete mechs, like Kerberos V
  - draft-ietf-nfsv4-ccm-02.txt
- Properly handles channel bindings proof exchanges
  - Establishes security context for concrete mech
  - Initiators prove channel bindings to acceptors and vice-versa
- Offering CCM-BIND ***signals*** <u>willingness to use channel bindings</u>

CCM-MIC

- GSS *pseudo*-mechanism (not stackable)
- Uses previously established, *live* CCM-BIND security contexts to establish CCM-MIC contexts (bound to the same channel)
- CCM-MIC security context establishment is cheaper than CCM-BIND
  - Uses only MICs from concrete mech stacked below CCM-BIND in the construction of CCM-MIC context tokens
- Aim: further perf improvements for NFS

SASL w/ Channel Bindings

- Use SASL GSS-API spec
- And use CCM-BIND
- Negotiate SASL mechanisms as usual
  - If CCM-BIND is selected then use channel bindings
  - Else don't
- SASL security layers for CCM-BIND are noop

SPNEGO and Channel Bindings

- Require use of SPNEGO mech-specific GSS extensions, GSS_Spnego_set/get_neg_mechs() [rfc2478]
  - App must explicitly request CCM-BIND this way and must pass channel bindings
- SPNEGO should not pass channel bindings to traditional mechs (see stackable mechs I-D, slides)
- Negotiate mechs as usual

Stackable GSS Pseudo-Mechs

- In designing CCM-BIND we noticed a pattern worth abstracting[*]: stackable pseudo-mechs
- Optional interfaces for “indicating” such mechs are needed
- Optional interfaces for inquiring mechs for/by “attributes” also look to be useful; see:
  - draft-williams-gssapi-stackable-pseudo-mechs-00.txt
  - Presentation at KITTEN BoF

Internet-Drafts

- draft-ietf-nfsv4-channel-bindings-02.txt
- draft-ietf-nfsv4-ccm-02.txt
- draft-ietf-ipsp-apireq-00.txt
- draft-williams-gssapi-channel-bindings-00.txt
  - (missed new I-D cut-off)
- draft-williams-gssapi-stackable-pseudo-mechs-00.txt

History

- 2003/02/25, 1st CCM I-D
- CCM -00 I-D led to 1st channel bindings I-D
  - Which led to discussion of channel bindings to IPsec
- First presented to SAAG at 58th IETF
  - Original IPsec channel bindings proposal proved controversial, flawed
  - Subsequently led to current channel bindings to IPsec proposal
- This and other work aroung the GSS-API led to the KITTEN BoF at this IETF meeting

Q/A

- Questions?
- Please review