Channel Bindings

draft-ietf-nfsv4-channel-bindings-00.txt

Nicolas.Williams@sun.com

NOTE: The slides have **not** been modified since the presentation made at the 58th IETF SAAG meeting, but **notes** have been added.

NOTE: One slide has been added – its title is marked with an asterix and colored red

ToC

Goals*

- To present the [not new] concept of **channel bindings**
- To present the concept of **IPsec channels**
  - and their use by apps that have app-layer authentication
  - and their use to provide authentication facilities to apps
  - and their construction
    - and the need for related interfaces
- To address the disconnect between names and addresses with IPsec channels

- This slide is new since the presentation at the 58th IETF SAAG meeting

Abstract

- Channel bindings allow applications that provide for authentication to prove that the end-points at the app-layer **are the same** as at another secure lower layer
- A concept from the GSS-API, but needed to be formalized
  - see RFC2743, section 1.1.6
  - draft-ietf-nfsv4-channel-bindings-00.txt

From RFC2743

- “ Specifically, [channel bindings] enable GSS-API callers to bind the establishment of a security context to relevant characteristics (e.g., addresses, transformed representations of encryption keys) of the underlying communications channel, of protection mechanisms applied to that communications channel, and to application-specific data.”

•The phrase “transformed representations of encryption keys” is rather vague; its formalization is a subject of this presentation and associated I-D

Why Channel Bindings

- CBs allow pushing of session crypto to lower layers, which:
  - Avoids double session crypto
  - Allows offloading of more processing to specialized HW (e.g., IP+IPsec+TCP/SCTP+DDP), thus boosting perf
- CBs are necessary for multi-user protocols, like NFSv4, that use RDDP/RDMA
  - w/o CBs the trade-off is perf. or session security
  - w/ CBs there is no perf. vs. security trade-off
- Benefits for NFSv4:
  - fewer live crypto contexts,
  - access to more common HW crypto acceleration
  - RDDP/RDMA w/ confidentiality and integrity protection

- One question at the meeting was [paraphrasing] “why can't NFS use a separate connection per-user?”
  - the first reason that comes to mind is that NFS uses ONC RPC, and ONC RPC doesn't currently sport support for the notion that all RPCs over a given connection are associated with the IDs authenticated by that connection
  - if applications could use IPsec for authentication that would be great, but we still need IPsec channels for that
  - IPsec needs to provide for the range of authentication technologies that applications and users use today if applications are to use IPsec is to be used by them for authentication
    - IPsec currently only does PKI and EAP, but the GSS-API and its mechs don't really fit into EAP – perhaps IKEv2 should have CERT/CERTREQ/AUTH types for use w/ the GSS-API

Channel Bindings Definition

- Channel Bindings are data that:
  - Is cryptographically derived from the key exchange/ negotiation of an established channel
  - Is the same on both end-points of the channel
  - **Cannot be made the same for two different channels by an attacker**
- E.g., the SSHv2 session ID is the CB data for SSHv2 channels; the concatenation of TLS finished messages forms the CB data for TLS

Contrived Example #1

- Kerberized TELNET over TLS
  - w/o channel bindings (but w/ address-less tickets)
- `telnet foo.bar.com`
  - connect(gethostbyname(“foo.bar.com”))
  - start TLS; start TELNET
  - krb5_get_cred(krb5_sname2princ(“host”, “foo.bar.com”))
  - send AP-REQ, wait for/verify AP-REP
  - KRB-SAFE/PRIV/krb5 session key **not used** at all
    - “ TLS will protect us”
- MITM can spoof DNS lookup, redirect the client to another server which speaks TLS and forward AP-REQ/REP between client and foo.bar.com...
  - MITM attack succeeds

•Eric Rescorla was correct about the application checking the server's TLS certificate: that defeats the MITM attack

•Jeff Altman was correct that this slide should have expressly referred to TLS w/ anon DH

•However, when supplanting IPsec for TLS the disconnect between the requested server name and the IPsec authentication of the server IP address becomes evident – I did not want to use IPsec in this example because at this point in the presentation the concept of IPsec channels had not come up yet.

- Thus the example is “contrived”

Contrived Example #2

- Kerberized TELNET over TLS
  - w/ channel bindings
- `telnet foo.bar.com`
  - connect(gethostbyname("foo.bar.com"))
  - start TLS; start TELNET
  - krb5_get_cred(krb5_sname2princ("host", "foo.bar.com"))
  - send AP-REQ, wait for/verify AP-REP
  - KRB-SAFE(TLS finished messages) exchange
    - CB match → no further use of KRB-SAFE/PRIV
- MITM can spoof DNS lookup, redirect the client to another server which speaks TLS and forward AP-REQ/REP..., **but cannot** make the channel bindings checks succeed → MITM attack is detected

- See previous slide's notes

But What About IPsec?

- Running NFSv4, iSCSI, ... over TLS is fine, but
- We want to run NFSv4, iSCSI, etc... over IPsec
- So what are IPsec channel bindings?
  - IPsec operates on IP packets; IP is connection-less
  - So what's an IPsec channel?!
    - no channel → no bindings
- So, define IPsec channels, and then the bindings to them.
  - IPsec channels allow for apps that let IPsec do authentication and session protection
  - Channel bindings to IPsec channels allow for apps that do authentication at the app-layer but let IPsec do their session protection

Strawman WARNING

- Details of TCP connection binding to IPsec and related interfaces are, if you don't like them, a strawman
  - regardless of whether the strawman stands or is torn to shreds one goal will have been accomplished

IPsec Channels

- An IPsec channel can be constructed as an extension to transports (TCP, SCTP, ...)
  - But no changes on the wire please
- TCP connection binding to IPsec bind to the transport mode IKEv2 IKE_SA that protects the **SYN**[1]
  - and to its re-key successors and children SAs,
  - and to any other SA that authenticates the same IDs, **if any**
    - and their re-key successors and children SAs
- [**1**] Protected SYN reception binds listener/acceptor
  - [**1**] Protected SYN|ACK reception binds initiator
  - Review pls!
- Recovery from expired certs/... is as from net partition

•IPsec channels are bound to a family of SAs that satisfy the related binding requirements, such as

- SAs must authenticate the same IDs as the initial binding, and/or
- SAs must be children SAs of the IKEv2 IKE_SA used for the initial binding, or of re-key successors of it

•The ID binding requirement works for **IKEv1** and **IKEv2**

•The IKE_SA binding requirement works only for IKEv2, but it allows for the use of unauthenticated IKE_SAs, which is useful for apps that provide for authentication at the app-layer so that they can use IPsec w/o having to have IPsec certs deployed – such applications must use channel bindings to protect against the MITM attack presented earlier

IPsec Channels Interfaces

- Before connecting/accepting connections, apps must specify IPsec binding requirements for their connections
- After connections are established apps must check their binding status
- If bound, apps that do provide for authentication at the app layer must then
  - retrieve the channel bindings for the connection
  - mutually authenticate the client and server
  - exchange and verify integrity-protected channel bindings
- Negotiation not needed if server always chooses binding
  - type of binding (integ. or conf.+integ.) must be specified
- If server does not want binding and client does see prev slide

- IPsec channels are all about interfaces
  - interfaces between applications and transports
  - and between transports and IP/IPsec
- IPsec channels may even work with connectionless transports, such as UDP, provided appropriate interfaces such that apps may enforce the binding requirement on datagrams sent and received as appropriate

In sockets API Terms

- Clients & servers setsockopt(s, IPSEC_BIND)
  - Before connect()/listen()
  - Specify binding reqs: integrity or conf.+integ. prot.
- Clients & servers getsockopt(s, IPSEC_BIND)
  - After connect()/accept()
  - Retrieve binding status (bound/not-bound)
  - Retrieve channel bindings data
- Clients & servers mutually authenticate using GSS-API, or whatever, and, in the process or afterwards, exchange and verify integ.-protected hashes of the CBs
- An option to **not** protect a connection would be good (e.g., for IKEv2 itself)
  - At least one implementation has such a socket option

- Socket options named here are purely fictitious – any resemblance to real-life socket options is purely accidental

Other IPsec Channels APIs?

- IPSEC_AUTH_AS (used w/ IPSEC_BIND)
  - Authenticate as given user or with given cert / cert name
  - Find out name / cert of self
- IPSEC_AUTH_TO (used w/ IPSEC_BIND)
  - Specify peer cert /cert name / acceptable CAs
  - Find out name/cert of peer
- I.e., enable apps to use IPsec for authentication and session protection

IPsec Channel Bindings

- The concatenation of the octets normally signed in AUTH payloads of IKEv2 (see section 2.15 of IKEv2) for the initial SA to which a connection was bound

Security and Other Considerations

- App-layer QoPs vs. lower layer QoPs
- IPsec channel construction
  - Relation to processing model in 2401bis
  - Negotiation of IPsec channel construction has problem when the client wants but the server doesn't
    - new SA mode (“binding mode SA”) would solve this
- Must servers prove to clients that the CBs match?
  - If so, then should the CB exchange be:
    - Integ(H(direction || CB))
  - Kerberos V GSS-API mech does not do this...
- Negotiation of channel bindings use (see CCM-BIND-*)
- Interfaces, APIs (IETF work? sock opts? or utility func.?)
- Spec SCTP binding to IPsec

•W.r.t. 2401/2401bis, application-requested IPsec channels can be seen as moving IPsec policy from the SPD to the apps

•The 4th and 5th bullet points in this slide refer to a possible problem w.r.t. assymetric policies on the initiator and responder sides

Alternatives?

- MITM attack in 1st contrived example can be foiled by the use of secure name services (e.g., /etc/hosts, DNSSEC), but:
  - the apps cannot really be sure that the name service is secure
  - SPDs can change, so apps have no connection binding guarantee
  - not very general
    - does not help NFSv4 w/RDDP

•Another alternative is to use hostnames as addresses for sockets – this addresses the disconnect between authentication of names, addresses and name service lookups just as much as using secure name services does, but in a way that is visible to applications

Uses of Channel Bindings

- NFSv* w/ or w/o RDDP
- iSCSI, w/ or w/o RDDP
- Anything that uses RDDP
- HIP?
  - HIP client daemon would open a bound connection to the HIP server daemon, then authenticate the HITs, verify channel bindings and use the resulting SA to bind the application's HIT-to-HIT connection
    - Thus HIP need not negotiate SAs, just leverage IKEv2
    - w/o channel bindings HIP has to be an SA negotiation protocol – ugh
- Even TELNET, even SSHv2, X11, etc...

Documents

- draft-ietf-nfsv4-channel-bindings-00.txt
- draft-ietf-ipsp-ipsec-apireq-00.txt (closely related)
- draft-ietf-nfsv4-ccm-02.txt (out of date, will be updated after MPLS)
  - Defines CCM-BIND-*, GSS-API pseudo-mechanisms for dealing with channel bindings and negotiation of their use
- RDDP I-Ds – see RDDP WG page
  - RDDP needs this because DDP/RDMA is generally inserted between the transport and the application and DDP must point into the app protocol data, which must be in the clear from DDP's p.o.v.
- RFC2743 section 1.1.6, and RFC2744 section 3.11

- RDDP needs to reference sections of the plaintext application data above it – the application data need not be in plaintext from RDDP's p.o.v. on the wire, but it helps in the design of high-performance RNICs if it is