Connection Latching

draft-ietf-btns-connection-latching-00.txt
Nicolas.Williams@sun.com
What

• Connection latching is to IPsec as TCP is to IP
  – Protect packet flows coherently, not just packets
  – Kind of like TCP builds byte streams out of packets
• Alternatively: a way to build “IPsec channels”
Why

• IPsec protects packets, not packet flows
  – But we want to protect flows
    • Policies that aggregate multiple nodes and allow them to claim addresses from a common network make it possible for them to steal each other's packet flows (see descriptions at IETF65/66)
    • Policies can change, leaving live packet flows unprotected, or protected differently than before

• A foundation for IPsec APIs
• IPsec channels please (for channel binding to IPsec, thanks)
How

• On “connection” creation trigger create latch
  – figure out what the peer's ID is, what kind of SA protected the incoming or outgoing trigger
  – record this somewhere
  – make sure that all subsequent packets for that packet flow are protected by similar SA
    • drop incoming packets that aren't
    • don't send packets if you can't
• On connection end tear down the latch
How

- I-D sketches two implementation designs

- Approach #1: record latch in ULP TCB, communicate incoming/outgoing packet SA params between IPsec layer and ULP
  - In datagram-oriented apps the latch would be recorded/enforced by the app

- Approach #2: record the latch in PAD/SPD
  - Enforcement at IPsec policy layer
Approach #1: “Intimate interfaces”

- ULPs and IPsec interface with ancillary data attached to packets as they move up/down the stack
  - U->I “tell me how you'll protect this outgoing packet”
  - I->U “this incoming packet was protected like so”
  - U->I “protect this packet like so”
  - Record latch in ULP TCB, enforce at ULP

- For UDP use “connected” sockets, else put the app in charge of recording/enforcing latch
Approach #2: PAD-based latching

• Listeners create 3-tuple “template” PAD entry
• Initiators create 5-tuple “template” PAD entry
• Packets that match a template PAD entry cause an actual PAD entry to be created
  – child SA constraints populated from the packet
  – peer ID populated from the SA that protected the incoming packet
• On connection tear-down the “cloned” PAD entry is removed
Properties

• Approach #1 works for connection-oriented and non-connection-oriented ULPs
  – For UDP apps can “connect” UDP sockets, **OR**
  – apps can record/enforce latch through “pTokens” on sendmsg()/recvmsg()

• Approach #2 only allows for “connected” UDP

• Approach #2 needs a TIME_WAIT-like state
  – Flow and latch tear down have to be atomic w.r.t. new flow triggers
  – Wait time given by local latencies only
Properties

• BUT

  – Approach #2 is very close to RFC4301 model and so can be used with NICs that provide ESP/AH/SPD offload but which don't also provide packet tagging interfaces needed for approach #1
APIs

- At its simplest traditional connect()/accept() BSD socket-type APIs can perform connection latching without the app even knowing
APIs

• But IPsec APIs can give apps more power
  – Who am I really talking to (IP addresses don't do)
  – Specify/verify that a connection's QoP meet/meets some QoP policy
  – LoF
  – etc...

• See Michael's and Miika's I-Ds/presentations
BYPASS OR PROTECT

• Oh yeah, optional (“opportunistic”?) protection
  • Motivation: if IPsec works, use it and channel binding, else do what the app always did (crypto at app layer)
    – In Approach #1 this is handled by the ULP or app
      • But a BYPASS OR PROTECT SPD entry is needed so that IPsec knows whether the ULP/app can handle this
    – In Approach #2 this requires an SPD extension
BYPASS OR PROTECT

- Approach #1 -> ULP/app is responsible, but a BYPASS OR PROTECT SPD entry still needed
- Approach #2 -> *template* SPD BYPASS OR PROTECT entry needed, similar to template PAD entry
  - When a matching packet arrives unprotected clone a BYPASS entry for just that 5-tuple
  - When a matching packet arrives protected clone a PROTECT entry for just that 5-tuple
  - ULP must tear these down when the flows end
  - On outgoing flow initiating packets the app must request protection or bypass, and it must handle IKE timeouts
**BYPASS OR PROTECT**

- Example: NFSv4 client could try to connect to server on default port (2049) using IPsec (BTNS OK)
  - If it works, use RPCSEC_GSS with GSS channel binding to the IPsec channel
  - Else (no IKE on server, or IKE timeout) try again w/o IPsec, then use RPCSEC_GSS as usual with integ or conf+integ protection
    - as it would today