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
- Alternative description: CL is a way to build "IPsec channels"

Why

- IPsec protects packets, not packet 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
  - 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 tear down end tear down the latch

OK, really, how

- Two implementation designs are sketched as examples
- 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
  - For UDP apps can record/enforce latch through “pTokens” on sendmsg()/recvmsg()
- Approach #2 does not allow for flexible app-driven latching for UDP
- Approach #2 needs a TIME_WAIT type state

Properties

- BUT, approach #2 is very close to RFC4301 model and so can be used with NICs that provide ESP/AH/SPD offload without providing packet tagging interfaces needed for approach #1

APIs

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

APIs

- See Michael's and Miika's I-Ds/presentations

Q/A

- BTW, there's at least one implementation using approach #1 + some APIs, but not enough
  - See ipsecconf(1M) and ipsec(7P) in Solaris
- KAME has an implementation as well (Michael: using which approach?)
- questions