Trust Management and Network Layer Security Protocols

Matt Blaze[1] and John Ioannidis[1] and Angelos D. Keromytis[2]

[1] AT&T Laboratories – Research
{mab,ji}@research.att.com
[2] Distributed Systems Labs
CIS Department, University of Pennsylvania
angelos@dsl.cis.upenn.edu

1 Introduction

Network-layer security among mutually trusting hosts is a relatively straightforward problem to solve. The standard protocol technique, employed in IPSEC [KA98], involves "encapsulating" an encrypted network-layer packet inside a standard network packet, making the encryption transparent to intermediate nodes that must process packet headers for routing, *etc.* Outgoing packets are authenticated, encrypted, and encapsulated just before being sent to the network, and incoming packets are decapsulated, verified, and decrypted immediately upon receipt[IB93]. Key management in such a protocol is similarly straightforward in the simplest case. Two hosts can use any key-agreement protocol to negotiate keys with one another, and simply use those keys as part of the encapsulating and decapsulating packet transforms.

In many applications, security at the network later has a number of advantages over security provided elsewhere in the protocol stack. Network semantics are usually hidden from applications, which therefore automatically and transparently take advantage of whatever network-layer security services their environment provides. Especially importantly, the network layer offers a remarkable flexibility not possible at higher- or lower- abstractions: security can be configured end-to-end (protecting traffic between two hosts), route-to-route (protecting traffic passing over a particular set of links), edge-to-edge (protecting traffic as it passes between "trusted" networks via an "untrusted" one), or in any other configuration in which network nodes can be identified as appropriate security endpoints.

Fortunately, the design of encapsulation techniques for basic authentication and confidentiality services is not a conceptually difficult problem, and network-layer security protocols, such as IPSEC, have matured to the point of being standardized and implemented by commercial vendors.

A harder problem, however, and one that current standards for network-layer security do not address, is the management of the policy governing the handling of packets on the way in to or out of a host running the encapsulation protocol. By itself, the security protocol protects packets from external tampering and eavesdropping, but does nothing to enforce a policy as to which hosts are authorized to exchange particular kinds of traffic. In many configurations, especially when network-layer security is used to build firewalls and virtual private networks, such polices can be quite complex.

Central to the problem of engineering policy mechanisms for network security is the tradeoff between expressiveness and performance. Unfortunately, many configurations demand a high level of both.

In this position paper, we examine the problem of managing policy in network-layer security protocols, and propose a trust-management architecture for network-layer security that may satisfy both the expressibility and the performance issues.

2 IPSEC Policy Architecture

Let us examine the architecture of network-layer security more closely, using IPSEC as a specific example. In this environment, policy must be enforced whenever packets arrive at or are about to leave a network security endpoint (which could be an end host, a gateway, a router, or a firewall).

When an incoming packet arrives from the network, the security endpoint first determines the processing it requires:

– If the packet is not protected, should it be accepted? This is essentially the "traditional" packet filtering problem, as performed, *e.g.,* by network firewalls.
– If the packet was encapsulated under the security protocol:
  - Is there correct key material (usually contained in a data structure called a "security association") required to decapsulate it?
  - Should the resulting packet (after decapsulation) be accepted? A second stage of packet filtering occurs at this point. Notice that a packet may be successfully decapsulated and still not be accepted (*e.g.,* a decapsulated packet might contain an illegal network source IP address such as 127.0.0.1).

A security endpoint makes similar decisions when an outgoing packet is ready to be sent:

– Is there a security association (SA) that should be applied to this packet? If there are several applicable SAs, which one should be selected?
– If there is no SA available, how should the packet be handled? It may be forwarded to some network interface, dropped, or queued until an SA is made available, possibly after triggering some automated key management mechanism such as the IPSEC ISAKMP protocol[HC98].

Observe that because these questions are asked on packet-by-packet basis, policy filtering must be performed, and any related security transforms applied, quickly enough to keep up with network data rates. This implies that in all but the slowest network environments there is insufficient time to process elaborate security languages, perform public key operations, consult large tables, or resolve rule conflicts in any sophisticated manner.

Implementations of network layer security services, including IPSEC and most firewalls, therefore, usually employ very simple, filter-based languages for configuring their packet-handling policies. In general, these languages specify routing rules for handling packets that match bit patterns in packet headers, based on such parameters as incoming and outgoing addresses and ports, services, packet options, *etc.*[MJ93]

However, packet-level filtering – necessary as it might be – is not the interesting problem.

3 Policy and Security Associations

A basic parameter of the packet processing problems mentioned in the previous section is the question of whether a packet falls under the scope of some Security Association (SA). SAs contain and manage the key material required to perform network-layer security protocol transforms. How then, do SAs get created?

The obvious approach involves the use of a public-key or Needham-Schroeder [NS78] based key distribution scheme as the basis for a protocol that creates a new SA with whatever host attempts to communicate unsecured traffic in a manner that fails the packet-level security policy. At least one currently-distributed IPSEC implementation does just this, with the aim of performing "opportunistic encryption" whenever possible.

Unfortunately, protocols that merely arrange for packets to be protected under security associations do nothing to address the problem of enforcing a *policy* regarding the flow of incoming or outgoing traffic. Recall that policy control is a central motivation for the use of network-layer security protocols in the first place.

In general, and rather surprisingly, security association policy is largely an open problem – one with very important practical security implications and with the potential to provide a solid framework for analysis of network security properties.

Fortunately, the problem of policy management for security associations can be distinguished in several important ways from the problem of filtering individual packets. In particular:

- SAs tend to be rather long-lived; there is "locality of reference" insofar as hosts that have exchanged one packet are very likely to also exchange others in the near future.
- It is acceptable for SA creation to require substantially more resources than can be expended on processing every packet (*e.g.,* public key operations, several packet exchanges, policy evaluation, *etc.*)
- The "output" of negotiating an SA between two hosts can provide (among other things) parameters for lower-level packet filtering operations.

A trust-management system[BFL96], such as *KeyNote*[BFK99], may be of value here.

4 A Trust Management Architecture for Network Layer Security

The problem of controlling SAs in a network-layer security protocol is easy to formulate as a trust-management problem. Trust-management systems are characterized by:

- A method for describing "actions," which are operations with security consequences that are to be controlled by the system.
- A mechanism for identifying "principals," which are entities that can be authorized to perform actions.
- A language for specifying application "policies," which govern the actions that principals are authorized to perform.
- A language for specifying "credentials," which allow principals to delegate authorization to other principals
- A "compliance checker," which provides a service for determining how an action requested by principals should be handled, given a policy and a set of credentials.

The trust-management approach has a number of advantages over other mechanisms for specifying and controlling authorization, especially when security policy is distributed over a network or is otherwise decentralized.

In the case of SA policy, the "actions" would represent the low-level packet filtering rules required to allow two hosts to conform one another's higher-level policies.

This suggests a simple framework for trust management for Network- Layer Security:

- Each host has its own trust-management-controlled policy governing SA creation. This policy specifies the classes of packets and under what circumstances the host will initiate SA creation with other hosts, and also what types of SAs it is willing to allow other hosts to establish.
- When two hosts discover that they require an SA, they each propose to one another the "least powerful" packet-filtering rules that would enable them to accomplish their communication objective. Each host sends proposed packet filter rules, along with credentials (certificates) that support the proposal. The trust structure of these credentials is entirely implementation dependent, and might include the arbitrary web-of-trust, globally trusted third-parties, or anything in between.
- Each host queries its trust-management system to determine whether the proposed packet filters comply with local policy and, if they do, creates the SA containing the specified filters.

Other SA properties might also be subject to trust management policy. For example, the SA policy might specify acceptable cryptographic algorithms and key sizes, the lifetime of the SA, logging and accounting requirements, *etc.*).

Our architecture divides the problem of policy management into two natural components: packet filtering, based on simple rules applied to every packet, and trust management, based on negotiating and deciding which such rules are trustworthy enough to install.

This distinction makes it possible to perform the per-packet policy operations at high data rates while effectively establishing more sophisticated trust-management-based policy controls over the traffic passing through a secure end-point. Having such controls in place makes it easier to specify security policy for a large network, and makes it especially natural to integrate automated policy distribution mechanisms.

An important practical problem in introducing security policy mechanisms is the transition from older schemes into the new one. Existing IPSEC security policies, which are based only on packet filters, quite easily fit into the trust management framework. As the trust management mechanism is introduced, filter-based policies can be mechanically translated into trust-management policies and credentials.

5 Conclusions and Status

We have developed a number of trust management systems, and have started examining the use of *KeyNote* in the engineering of network-layer security protocols. We are in the process of implementing an IPSEC architecture similar to that described above; it is our hope that the formal nature of trust management will make possible network security configurations with provable properties. One of the most relevant features of trust management to SA management is the handling of policy delegation.

Furthermore, because KeyNote is application-independent, it can be used to "tie together" different aspects of network security, beyond just IPSEC and packet filtering. For example, a more comprehensive network security policy could specify what mechanisms are acceptable for remote access to a private corporate network over the Internet; such a policy might, for example, allow the use of cleartext passwords only if traffic is protected with IPSEC or some transport-layer security protocol (*e.g.,* SSH [YKS+99]). Multi-layer policies would, of course, require embedding policy controls into either an intermediate security enforcement node (such as a firewall) or into the end applications.

Finally, if trust-management policies and credentials are built into the network security infrastructure it may be possible to use them as an "intermediate language" between the low-level application policy languages (*e.g.,* packet-filtering rules) and higher-level policy specification languages and tools. A translation tool would then be used to convert the high-level specification to the trust-management system's language (and perhaps vice-versa as well). Such a tool could make use of formal methods to verify or enforce that the generated policy has certain properties.

There are many open, and we believe, quite interesting and important problems here.

References

[BFK99] M. Blaze, J. Feigenbaum, and A. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In *Proceedings of the 1998 Cambridge Security Protocols International Workshop*, pages 59–63. Springer, LNCS vol. 1550, 1999.

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[HC98] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[IB93] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network-Layer Security Under Unix. In *Fourth Usenix Security Symposium Proceedings*. USENIX, October 1993.

[KA98] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Request for Comments (Proposed Standard) 2401, Internet Engineering Task Force, November 1998.

[MJ93] Steven McCanne and Van Jacobson. A BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the Annual USENIX Technical Conference*, pages 259–269, San Diego, California, January 1993. Usenix.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–998, December 1978.

[YKS$^+$99] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Draft, Internet Engineering Task Force, February 1999. Work in progress.