Managing Access Control in Large Scale Heterogeneous Networks

Angelos D. Keromytis, Kostas Anagnostakis, Sotiris Ioannidis, Michael B. Greenwald and Jonathan M. Smith

***Abstract*— The design principle of maximizing local autonomy except when it conflicts with global robustness has led to a scalable Internet with enormous heterogeneity of both applications and infrastructure. These properties have not been achieved in the mechanisms for specifying and enforcing security policies.**

The STRONGMAN (for Scalable TRust Of Next Generation MANagement) system [9], [10] offers three new approaches to scalability, applying the principle of *local* policy enforcement complying with *global* security policies. First is the use of a compliance checker to provide great local autonomy within the constraints of a global security policy. Second is a mechanism to compose policy rules into a coherent enforceable set, *e.g.,* at the boundaries of two locally autonomous application domains. Third is the "lazy instantiation" of policies to reduce the amount of state that enforcement points need to maintain.

In this paper, we focus on the issues of scalability and heterogeneity.

I. Introduction

Internet security is a rapidly growing challenge. Damage of computing resources (*e.g.,* hard drives erased) is only one example of possible malice. Increases in both the number and heterogeneity of systems attached to the net worsens the problem, as techniques to manage this scale and complexity have been slow to emerge. Many aspects of network design and implementation have managed to scale well, mainly as a byproduct of intelligent application of the end-to-end design principle ([12], [4]), which states that properties that must hold end-to-end are provided by mechanisms at the end points. Application of this principle has kept the network simple and allowed great autonomy in implementing these mechanisms. Unfortunately, security has not benefited from this philosophy, despite its end-to-end nature.

By the end-to-end argument, hosts should be responsible for the perceived security of the network at large. However, the prevalence end-to-end *un-friendly* components, such as firewalls and Virtual Private Networks (VPNs), serves as proof that other factors come into play. We argue that three such factors, all related to an organization's security posture (and policies) have played an important role in forcing security functionality inside the network infrastructure.

1) An organization's policies must typically be specified at the granularity of administrative domains (*e.g.,* a corporate network), and not only at the granularity of individual hosts. This becomes difficult to implement as the size of a typical organization (in terms of workstation and other networked computing elements) increases.
2) Some operating systems have been designed under the assumption that network security is mostly handled by third parties (firewalls), thus lacking enforcement mechanisms. While this is a second-order effect of the current state of affairs in security design, it creates enough inertia against change towards a more architecturally sound approach.
3) Many security policies adopt the "hard shell, soft interior" approach, by granting more rights to "local" (and, by implication, trusted) machines and entities. Intuitively, we would expect, *e.g.,* employees of an organization to have greater access to that organization's network and data than outsiders — which is in fact what these security policies attempt to express. However, such designs are taken to extremes and result in such a "soft interior" (in terms of internal defense mechanisms) that, not only malicious insiders become the most important threat, but the increased level of connectivity of organizations results in a race between mechanisms such as firewalls on the one hand and network connectivity on the other, in very unfavorable terms for the defense.

Consider, for example, the pervasive use of firewalls, which enforce a single security policy at network boundaries to protect multiple hosts behind the boundaries from certain classes of security problems. To implement the policy globally, the network topology must be restricted to pass all traffic through the firewall. Apart from a firewall's negative consequences for Internet routing, flow control, and performance, when the firewall fails or is otherwise bypassed, the entire internal network is at the mercy of the intruder. Traditional firewall work has focused on nodes and enforcement mechanisms rather than overall network protection and policy coordination. Thus, in some sense, security mechanisms and access control have interfered with simplicity and scalability.

Any alternative that attempts to avoid the performance bottleneck of a centralized firewall must support a simple (and *consistent*) specification of security policy for an entire administrative domain. In other words, there must be means of ensuring that the local enforcement actually conforms to the larger ("global") policy. Since manual or semi-automatic configuration of nodes and protocols to conform to a global policy has been shown to be problematic and error-prone [7], automatic techniques relying on a single method of specification are desirable.

It may seem natural to generalize the solution proposed by distributed firewalls ([2], [8]) or other similar approaches [1], [6], [5], [11] and design a "universal" high-level policy specification language. Such a language would, ideally, specify global policies that must be enforced across multiple heterogeneous domains. However, security policies are often application-dependent. "Universal" high-level policy lan-

This work was supported by DARPA and NSF under Contracts F39502-99-1-0512-MOD P0001 and CCR-TC-0208972 respectively.

Angelos D. Keromytis is with the CS Department, Columbia University, Email: angelos@cs.columbia.edu. Kostas Anagnostakis, Sotiris Ioannidis, Michael B. Greenwald, Jonathan M. Smith are with the CIS Department, University of Pennsylvania, Email: {anagnost,sotiris,mbgreen,jms}@dsl.cis.upenn.edu

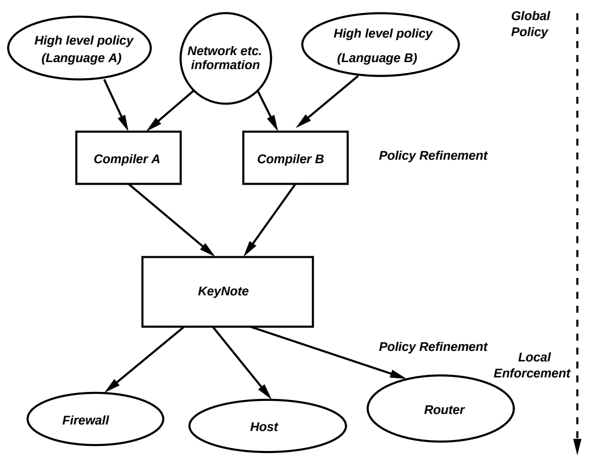

Fig. 1. **KeyNote used as a policy interoperability layer. Policy composition in STRONGMAN does not depend on using the same compiler to process all the high-level policies.**

guages tend to be feature-rich and complex, and are therefore clumsy and lead to mistakes. Furthermore, such languages often presume homogeneity, and cannot handle mixtures of multiple mechanisms/languages for different parts of the same network.

We argue that the correct approach is an architecture that ties together multiple security mechanisms within a single system image, that supports *many* application-specific policy languages, that automatically distributes and uniformly enforces the single security policy across all enforcement points, and that allows enforcement points to be chosen appropriately to meet both security and performance requirements. Further, this architecture must scale with the growth of the network in several dimensions (number of users, hosts, protocols/applications, and security policies tying all these together).

Our STRONGMAN Architecture [10] addresses these requirements. The main components of our architecture are the use of a policy compliance checker to provide great local autonomy within the constraints of a global security policy, a mechanism for composing policy rules into a coherent enforceable set, and "lazy instantiation" of policies to reduce the amount of state that enforcement points need to maintain. While we have described the STRONGMAN architecture at length elsewhere, here we pay particular attention to scalability and heterogeneity issues first analyzed in Keromytis [9].

II. The STRONGMAN Architecture

Following our previous discussion, we have set certain requirements for our proposed system. First, it must handle growth in the number of users, applications, enforcement points, and rules pertaining to these. A corollary to this is that the most common operations (*i.e.,* policy updates) must be very cheap. Second, security policies for a particular application should be specifiable in an application-specific language or application-specific extension. Third, administrators should be able to independently specify policies over their own domain: this should be true whether the administrator manages particular applications within a security domain, or manages a sub-domain of a larger administrative domain. In other words, the system must support privilege delegation and hierarchical management.

These requirements shape our design of the STRONGMAN architecture. An overview of the policy flow in our architecture is shown in Figure 1. It should be immediately clear that there is a distinction between high and low level policy. In particular, we envision a multiplicity of high-level policy specification mechanisms (different languages, GUIs, *etc.*), all translating to the same lower-level policy expression language. A powerful, flexible, and extensible low-level mechanism that is used as a common "policy interoperability layer" allows us to use the same policy model across different applications, without mandating the use of any particular policy front-end. This architecture has an intentional resemblance to the IP "hourglass", and resolves heterogeneity in similar ways, *e.g.,* the mapping of the interoperability layer onto a particular enforcement device, or the servicing of multiple applications with a policy *lingua franca*.

As the figure also implies, policy is enforced in a decentralized manner. STRONGMAN shifts as much of the operational burden as possible to the end users' systems because traditional enforcement points are generally overloaded with processing requests and mediating access. In our architecture, we can have an arbitrary number of enforcement points, deployed at the granularity necessary to enforce very fine-grained access control. This, however, can lead to excessively large numbers of policy rules (in the worst case, the cross-product of the number of users, number of nodes, and number of services per node). In order to minimize the resources consumed by policy storage and processing at each enforcement point, the low-level policy system supports "lazy instantiation" of policy. In other words, an enforcement point should only learn those parts of the global policy that it actually has to enforce as a result of user service access patterns. A further benefit of this approach is that policy may be treated as "soft state," and thus be discarded by the enforcement point when resources are running low, and recovered when space permits or after a crash.

Other important aspects of our architecture, not shown in Figure 1, include:

- Independent policy specifications can be composed in a manner which does not violate any of them, because multiple independently-specified policies may be managed at a single enforcement point.
- Users are identified by their public keys (each user may have multiple keys, for different purposes/applications). These public keys are used in the context of various protocols to authenticate the users to specific services. This also helps prevent malicious users from tampering with policies provided to enforcement points via "lazy policy instantiation".
- The low-level policy system allows for decentralized and hierarchical management and supports privilege delegation to other users. Note that delegation allows any user to be treated as an "administrator" of her delegatees; conversely, administrators in such a system can simply be viewed as users with very broad privileges.

```
permit KEY1 if
    using strong encryption and
    target in 192.168.1.0/24

permit USERGROUP4 if
    using authentication and
    origin in LOCALNET and
    target in WEBSERVERS
```

Fig. 2. **A high-level IPsec policy, enforced at the network layer.**

This permits both decentralized management (different administrators/users are made responsible for delegating and potentially refining different sets of privileges), and collaborative networking (by treating the remote administrator as a local user with specific privileges she can then delegate to her users). Limited privileges can be conferred to administrators of other domains, who can then delegate these to their users appropriately; this allows for Intranet-style collaborations.

Our architecture implements these design principles by using the KeyNote [3] trust-management system as a basis for expressing and distributing low-level security policy. In the next few subsections we give an overview of KeyNote, describe the policy translation and composition mechanisms, and discuss how policy is distributed (and how "lazy instantiation" is implemented) in our system.

A. KeyNote

KeyNote is a simple trust-management system and language developed to support a variety of applications. Although it is beyond the scope of this paper to give a complete tutorial or reference on KeyNote syntax and semantics (for which the reader is referred to [3]), we review a few basic concepts to give the reader a taste of what is going on.

The basic service provided by the KeyNote system is *compliance checking;* that is, checking whether a proposed *action* conforms to local *policy.* Actions in KeyNote are specified as a set of name-value pairs, called an *Action Attribute Set.* Policies are written in the KeyNote *assertion language* and either accept or reject action attribute sets presented to it. Policies can be broken up and distributed via *credentials*, which are signed assertions that can be sent over a network and to which a local policy can defer in making its decisions. The credential mechanism allows for complex graphs of trust, in which credentials signed by several entities are considered when authorizing actions. Users have a variety of credentials, for the different services and nodes they need to access.

Each service that needs to mediate access, queries its local compliance checker on a per-request basis (what constitutes a "request" depends on the specific service and protocol). The compliance checker can be implemented as a library that is linked against every service, as a daemon that serves all processes in a host, or as a network service (this latter case requires provisions for secure communications between the policy enforcer and the compliance checker).

B. Policy Translation and Composition

In STRONGMAN, policy for different network applications can be expressed in various high-level policy languages or

```
allow USERGROUP5 if file "/foo/bar.html"

allow ANGELOS if
    directory "/confidential" and
    source in LOCALNETWORK
```

Fig. 3. **A high-level web access policy, enforced by the web server.**

systems, each fine-tuned to the particular application. Each such language is processed by a specialized compiler that can take into consideration such information as network topology or a user database and produces a set of KeyNote credentials. At the absolute minimum, such a compiler needs a knowledge of the public keys identifying the users in the system. Other information is necessary on a per-application basis. For example, knowledge of the network topology is typically useful in specifying packet filtering policy; for web content access control, on the other hand, the web servers' contents and directory layout are probably more useful. Our proof-of-concept languages (examples are shown in Figures 2 and 3) use a template-based mechanism for generating KeyNote credentials.

This decoupling of high and low level policy specification permits a more modular and extensible approach, since languages may be replaced, modified, or created without affecting the underlying system. Heterogeneity is achieved by the design with its approach of *localized adaptation*. Note that the translation from KeyNote to architecture-specific formats allows new types of machines to be used in the overall architecture by building a single adaptor, that is, the one which converts KeyNote to the local format. Thus, if a new device is added to the system, it is a part of the access control enforcement system once KeyNote is able to be translated to its device-specific access control enforcement mechanism. Likewise, the translation from application-specific *policy languages* to KeyNote's common representation of access control policy allows many new distributed applications to be built with user-comprehensible domain-specific policy languages. It is our belief that almost any access control semantics can be captured within this framework.

Our architecture requires each high-level language or GUI to include a "referral" primitive. A referral is simply a reference to a decision made by another language/enforcement point (typically lower in the protocol stack). This primitive allows us to perform policy composition at enforcement time; decisions made by one enforcement mechanism (*e.g.*, IPsec) are made available to higher-level enforcement mechanisms and can be taken into consideration when making an access control decision. An example of this is shown in Figure 4. The only needed coordination between two policy domains is determining what kind of information (encoded in the referrals) needs to be generated and consumed respectively.

To complete the composition discussion, all that is necessary is a channel to propagate this information across enforcement layers. In our system, this is done on a case-by-case basis. For example, IPsec information can be propagated higher in the protocol stack by suitably modifying the Unix `getsockopt(2)` system call; in the case of a web server and SSL, the information is readily available through the SSL data structures (since the SSL and the web access control enforcement are both done in the context of a single process address space). This approach is sufficient for policy interaction across network layers, but would not work for arbitrary policy domain interaction.

C. Credential Management

Following our design decision of shifting as much as possible of the operational burden away from the enforcement points and to the users' systems, we make the users responsible for presenting the necessary credentials to the enforcement points they access. Thus, the enforcement points dynamically "learn" those parts of the global policy that are relevant to a particular request. It is in the interest of the user to present the correct credentials, in order to obtain service.

Compiled credentials are available to users through policy repositories. These credentials are signed by the administrator's key and contain the various conditions under which a specific user (as identified by her key in the credential) is allowed to access a service. The translation of the policy rule in Figure 4 is shown in Figure 5.

Users who wish to gain access to some service first need to acquire a fresh credential from one of the repositories. It is not necessary to protect the credentials as they are transferred over the network, since they are self-protected by virtue of being signed[1]. Users then provide these credentials to the relevant service (web server, firewall, *etc.*) through a protocol-specific mechanism. For example, in the case of IPsec, these credentials are passed on to the local key management daemon which then establishes cryptographic context with the remote firewall or end system. In the case of firewalls in particular, the user's system can either depend on a signaling mechanism (as is being developed at the IETF IP Security Policy Working Group) to detect their existence, or can statically analyze the KeyNote credentials to determine what actions need to be taken when trying to access specific services, networks, or end-systems.

It is also possible to pass KeyNote credentials in the TLS/SSL protocol. For protocols where this is not possible (*e.g.*, SSHv1), an out-of-band mechanism can be used instead. We have used a simple web server script interface for submitting credentials to be considered in the context of an access control decision; credentials are passed as arguments to a CGI script that makes them available to the web server access control mechanism. To avoid DoS attacks, entries submitted in this manner are periodically purged (in an LRU manner).

Since policy is expressed is terms of credentials issued to users, it need not be distributed synchronously to the enforcement points. As noted above, enforcement points do not need to store all credentials and rules; rather, they learn rules through "lazy policy instantiation" as users try to gain access to controlled resources. If needed credentials were discarded because of resource scarcity, the affected users will simply have to re-submit them with their next access.

[1]It is possible to provide credential-confidentiality by encrypting each credential with the public key of the intended recipient.

```
allow USER_ROOT if
  directory "/confidential" and
  source in LOCALNETWORK and
  (application IPsec says
         "strong encryption" or
   application SSL says
         "very strong encryption")
```

Fig. 4. **Web access policy taking into consideration decisions made by the IPsec and SSL protocols. The information on USER_ROOT and LOCALNETWORK are specified in separate databases, which the compiler takes into consideration when compiling these rules to KeyNote credentials.**

```
Authorizer: ADMINISTRATOR_KEY
Licensees: USER_ROOT_KEY
Conditions: app_domain == "web access" &&
  directory ~= "^/confidential/.*" &&
  (source_address <= "192.168.001.255" &&
   source_address >= "192.168.001.000") &&
  (ipsec_result == "strong encryption" ||
   ssl_result ==
     "very strong encryption");
Signature: ...
```

Fig. 5. **Translation of the policy rule from Figure 4 to a KeyNote credential. The public keys and the digital signature are omitted in the interests of readability.**

Adding a new user or granting more privileges to an existing user is simply a matter of issuing a new credential (note that both operations are equivalent). The inverse operation, removing a user or revoking issued privilege, can be more expensive: in the simple case, a user's credentials can be allowed to expire; this permits a window of access, between the time the decision is taken to revoke a user's privileges and the time the relevant credentials expire. For those cases where this is adequate, there is no additional overhead. This argues for relatively short-lived credentials, which the users (rather, software on their systems) will have to re-acquire periodically. While this may place additional burden on the repositories, it is possible to arrange for credentials to expire at different times from each other, thus mitigating the effect on the infrastructure of multiple users (re-)acquiring their credentials at the same time, if the credentials are relatively long-lived. Given that a large number of digital signatures will have to be computed as a result of periodically issuing credentials, this is also desirable from a policy-generation point of view.

For more aggressive credential revocation, other mechanisms have to be used. Although no single revocation mechanism exists that can be used in all possible systems, we note that any such mechanism should not increase the load or storage requirements on enforcement points. Thus, the most attractive approach is proofs of validity (acquired by the user from a "refresher" server, and provided to the enforcement point along with the credentials). The proofs of validity can be encoded as KeyNote credentials that are injected in the delegation chain. While this approach is architecturally attractive, it places high load on the refresher servers. The validity verification mechanism may be specified on a per-credential basis, depending on the perceived risk of compromise and the potential damage done if that occurs.

Finally, since KeyNote allows arbitrary levels of delegation (through chains of credentials), it is possible for users to act as lower-level administrators and issue credentials to others. In this way, we can build a hierarchical and decentralized management scheme wherein the corporate network administrator authorizes branch administrators to manage their networks under some constraints. More interestingly, it is possible to view the administrator of another network as a local user; that administrator can handle access to the shared resources for the remote network users, under the constraints specified in their credential, making easy the formation of so-called "extranets."

III. The Distributed Firewall

To validate our design choices and experiment with the different aspects of our architecture, we built a prototype distributed firewall. A distributed firewall (as described in [8]) enforces a single central security policy at *every* endpoint. The policy specifies what connectivity, both inbound and outbound, is permitted. This policy is distributed to all endpoints where it is authenticated and then enforced, thus making security an end-to-end property.

Distributed firewalls do not rely on the topological notions of "inside" and "outside" as do traditional firewalls. Rather, a distributed firewall grants specific rights to machines that possess the credentials specified by the central policy. A laptop connected to the "outside" Internet has the same level of protection as does a desktop in the organization's facility. Conversely, a laptop connected to the corporate net by a visitor would not have the proper credentials, and hence would be denied access, even though it is topologically "inside."

In the example STRONGMAN-based distributed firewall, endpoints are characterized by their public keys and the credentials they possess. Thus, the right to connect to the `http` port on a company's internal Web server is only granted to those machines having the appropriate credentials, rather than those machines that happen to be connected to an internal wire. With the advent of wireless LANs, such considerations are becoming extremely relevant.

In our prototype, end hosts (as identified by their IP address) are also considered principals when IPsec is not used to secure communications. This allows local policies or credentials issued by administrators to specify policies similar to current packet-filtering rules. Such policies or credentials have no option but to implicitly trust the validity of an IP address as an identifier. In that respect, they are equivalent to standard packet filtering. The only known solution to this is the use of cryptographic protocols to secure communications.

We should point out that the notions of a traditional and a distributed firewall are not incompatible. Traditional firewalls have an advantage over the distributed firewall in that they offer convenient aggregation points for network traffic, on which services such as denial of service detection (or, more generally, intrusion detection) are easier to deploy and operate. Furthermore, a combination of traditional and distributed firewalls offers "defense in depth", a well-established principle in physical security and the military world.

While space is limited, we can make the core scaling observation here. First, the approach of localized enforcement means that the number of enforcement points can be scaled to an arbitray network. Second, the late-binding [9] of policies to the enforcement points combined with network locality properties means that coordination traffic is limited. Finally, tree topologies can be used to scalably disseminate policy updates.

IV. Concluding Remarks

STRONGMAN is a new approach to security policy management. Its approach to scaling is local enforcement of global security policies. The local autonomy provided by compliance checking permits the architecture to scale comfortably with the Internet infrastructure. Our architecture accommodates considerable heterogeneity in both policies and enforcement points: the policy compliance checker composes policy rules into a coherent enforceable set for each enforcement point, and lazy instantiation reduces the state required at enforcement points. The removal of topological constraints in firewall placement facilitates other Internet protocols and mechanisms.

This paper has shown STRONGMAN's strengths in supporting scalable access control services in large heterogeneous networks. Security enforcement is pushed to the endpoints, consistent with end-to-end design principles. Since the enforcement points are coupled only by their use of a common global policy, they possess local autonomy which can be exploited for scaling. Our continued work in the architecture is focusing on both extending the security mechanisms that can be managed, and on providing administrators with intuitive, high-level application-specific languages and other management abstractions.

References

[1] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: a novel firewall management toolkit. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 17–31, May 1999.

[2] S. M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, November 1999.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.

[4] D. D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. SIGCOMM 1988*, pages 106–114, 1988.

[5] J. D. Guttman. Filtering Postures: Local Enforcement for Global Policies. In *IEEE Security and Privacy Conference*, pages 120–129, May 1997.

[6] S. Hinrichs. Policy-Based Management: Bridging the Gap. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.

[7] J. D. Howard. *An Analysis Of Security On The Internet 1989 - 1995*. PhD thesis, Carnegie Mellon University, April 1997.

[8] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS)*, pages 190–199, November 2000.

[9] A. D. Keromytis. *STRONGMAN: A Scalable Solution to Trust Management in Networks*. PhD thesis, University of Pennsylvania, December 2001.

[10] A. D. Keromytis, S. Ioannidis, M. Greenwald, and J. Smith. The STRONGMAN Architecture. In *Proceedings, DARPA Information Survivability Confernce and Exhibition*, volume 1, pages 178–188. IEEE Press, April 2003.

[11] A. Molitor. An Architecture for Advanced Packet Filtering. In *Proceedings of the 5th USENIX UNIX Security Symposium*, June 1995.

[12] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.