Path-based Access Control for Enterprise Networks[⋆]

Matthew Burnside and Angelos D. Keromytis

Computer Science Department
Columbia University
{mb,angelos}@cs.columbia.edu

Abstract. Enterprise networks are ubiquitious and increasingly complex. The mechanisms for defining security policies in these networks have not kept up with the advancements in networking technology. In most cases, system administrators define policies on a per-application basis, and subsequently, these policies do not interact. For example, there is no mechanism that allows a web server to communicate decisions based on its ruleset to a firewall in front of it, even though decisions being made at the web server may be relevant to decisions at the firewall. In this paper, we describe a path-based access control system for service-oriented architecture (SOA)-style networks which allows services to pass access-control-related information to neighboring services, as the services process requests from outsiders and from each other. Path-based access control defends networks against a class of attacks wherein individual services make correct access control decisions but the resulting global network behavior is incorrect. We demonstrate the system in two forms, using graph-based policies and by leveraging the KeyNote trust management system.

Key words: path-based, access control, Keynote, SOA, enterprise

1 Introduction

Most enterprise networks are distributed structures with multiple administrative domains and heterogeneous components. Defining and enforcing security policies in these networks is challenging – it is difficult for a system administrator or group of system administrators to conceptualize the security policy for such a network, let alone correctly express that policy in the myriad of languages and formats required by such an environment.

Consider a system where an oracle responds to all policy requests from the network. The complete, high-level policy for the network, as defined by the system administrator, is stored in and evaluated at the oracle. For every policy decision, a service queries the oracle and acts based on its response. Such a system

[⋆] This work was partially supported by NSF Grant CNS-07-14647 and by ONR MURI Grant N00014-07-1-0907. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the U.S. Government.

provides a globally coherent policy, but clearly does not scale well. Therefore, it is common practice to derive from the system administrators' high-level conceptual policy a set of policy components where each component is deployed at a single service or node. Each policy component is translated into the appropriate language for the target service and deployed directly at that service. In most cases, this task is performed by hand by the system administrator, though there have been some attempts at automating it, as in [1][2].

1.1 Example

Fundamentally, there is a violation of assumptions that comes from taking a high-level conceptual policy and componentizing it, either manually or mechanically. Consider the simple e-commerce network in Fig. 1. A firewall protects several hosts. On the first host are a web server and some business logic in the form of, *e.g.*, PHP or ColdFusion; on the second host is a database.

We propose a high-level conceptual policy for this network: all connections should arrive at port 80 on the firewall, authenticate at web server with a username and password, and the business logic must authenticate to the database using a public-key pair.

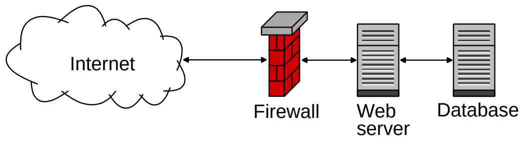

Fig. 1. A simple network. A web server and database are connected to the Internet through a firewall.

The system administrator derive from that high-level conceptual policy a set of policy components. One policy component is the firewall ruleset which blocks traffic to all ports except TCP port 80. Another component is the `.htaccess` file on the web server indicating that only a set of username/passwords may access the files containing the business logic. A final component is the grant table at the database which indicates that only the key pair used by the business logic may access the tables for that application. It is our contention that, in the process of generating these policy components, *path* information has been lost.

Consider an unknowing or malicious employee who plugs in a wireless access point, as in Fig. 2. An adversary can connect to this wireless access point and, through it, probe the web server and database. Such a connection violates the conceptual policy determined by the system administrator, but none of the individual policy mechanisms in place will detect it. That is, none of the policy mechanisms (the firewall ruleset, the `.htaccess` file, *etc.*)allows for governing how a request arrived at the service, but only what it requests after arrival.

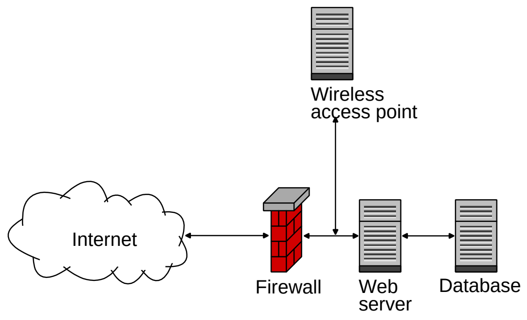

Fig. 2. A vulnerable network.

Similar flaws may occur if, for example, the firewall accidentally fails open due to misconfiguration or routing changes, or if an adversary attempts to access the business logic through some routing path that was unintentionally enabled. A compromised internal machine may be used to probe the remainder of the network. A misconfigured router may allow connections to bypass a firewall.

1.2 Contributions

In this work, we dynamically model the paths that requests take as they traverse an enterprise network and use those models as the basis for informing policy decisions. Requests traversing invalid paths are barred from penetrating deeper into the network. In a service-oriented architecture (SOA), the type of network we focus on in this paper, the path of interaction followed by a request is a tree. The root of the tree is the first point of interaction with the network (the firewall in the example above) and the branches of the tree represent the various actions taken by the various services in the network in response to that request. Enforcing policy consists of examining each pathway to determine that it followed a proscribed route.

We use a binary view in the policy-enforcement mechanism. Either a policy-proscribed service is in a pathway, or it is not. However, we also collect additional fine-grain details about events in each pathway in order to perform aggregate analysis. By exposing more information to downstream services, it is possible for the policy engine at each service to make decisions based on historical data or statistical trends. Note that the statistical analysis is not the focus of this paper and we do not address it further.

Accumulating path-traversal information benefits an enterprise by providing a simple, low cost, mechanism for preventing attacks that violate system administrators' assumptions about allowed or valid pathways. For example, a rogue wireless point is no longer the danger it once was. A misconfigured firewall will be more easily detected and many attacks during the window of vulnerability will be prevented.

In this paper, we present two solutions. The first is a low-cost, high-performance system that models incoming requests as graphs, where events are vertices and dependencies are edges. The second extends this concept and provides protection in some situations where internal nodes are untrusted; it leverages the KeyNote trust management system[3][4]. We show that in both cases, the performance overhead on the SOA is low.

The remainder of this paper is organized as follows. In Sect. 2, we discuss related work in the field. In Sect. 3, we describe the architecture of our two solutions. In Sect. 4, we give details on their implementation. We evaluate the work in Sect. 5 and conclude in Sect. 6.

2 Related Work

Most prior work in the policy field can be divided into three major categories: policy specification [3, 5], resolving policy conflicts [6, 7], and distributed enforcement [8, 9].

In their work in the field of trust management, Blaze, *et al.*, [10–12] built PolicyMaker, a tool that takes a unified approach to describing policies and trust relationships in enterprise-scale networks by defining policies based on credentials. It is based on a policy engine that identifies whether some request r with credentials c complies with policy p. In PolicyMaker, policies are defined by programs evaluated at runtime. SPKI [13–15] is a similar mechanism that uses a formal language for expressing policies. However, in both cases, the focus is on trust management rather than policy correctness. Both systems can be used as components in facilitating path-based access control, but alone, they are insufficient.

When there are multiple policies or multiple users defining policy there is always the possibility of conflict. Cholvy, *et al.* [7] describe a method for resolving that inconsistency and show that the problem is exacerbated in large-scale networks. As with PolicyMaker and SPKI, this method may facilitate path-based access control but it does not provide the information transfer necessary for resolving violated system administrator assumptions.

The STRONGMAN trust management system [2] focuses on the problem of scaling the enforcement of security policies and resolving policy conflicts. In STRONGMAN, high-level, abstract security policies are automatically translated into smaller components for each service in the network. STRONGMAN features no provision for future interaction between components, a key feature of path-based access control.

Bonatti, *et al.*, [16] propose an algebra for composing heterogeneous security policies. This is useful in networks with multiple policies defined in multiple languages (*i.e.*, most networks today). However, this system requires that *all* policies and supporting information and credentials be available at a single decision point, *e.g.* an oracle as discussed previously.

Firewalls [17, 18] are one of the most common and most well-known mechanisms for policy enforcement. The Firmato system [19] is a firewall management toolkit for large-scale networks. It provides a portable, unified policy language, independent of the firewall specifics. Firewall configuration files are generated automatically from the unified global policy. Firmato is limited to packet filtering, and it does not provide for future interactions between components.

The Oasis architecture [20] takes a wider view and uses a role-based system where principals are issued names by services. A principal can only use a new service on the condition that it has already been issued a name from a specific other service. Oasis recognizes the need to coordinate the dependencies between services, but since credentials are limited to verifying membership in a group or role, it is necessary to tie policies closely to the groups to which they apply.

In [21], the authors use KeyNote to distribute firewall rulesets, allowing endpoint nodes to perform enforcement independently. The path-based access control mechanism can be viewed as an extension of the distributed firewall system, allowing each endpoint node to incorporate the path of the request into its policy evaluation. The path-based system can further be viewed as an instantiation of the virtual private services described in [22]. Each request is presented a view of the network (a "private service") that is customized, based on the path the request has taken up to that point.

3 Architecture

In this section, we describe two methods for implementing path-based access control. The systems differ primarily in the mechanism by which the policy is evaluated. In the first system, we model policies and incoming requests as graphs, and we evaluate the policies by comparing the graphs representing actual requests with the policy graphs. The second system extends the first by modeling incoming requests and policies as KeyNote assertion chains.

Both systems are designed for use in SOA-style networks, so policy definition consists of defining trees representing valid requests. The policy distributed to each service is a list representing the path from the root to that service in the policy graph. This technique is simple and can be performed quickly, making it a good fit for dynamic networks where request patterns change quickly.

In both systems, the threat we consider is one where an adversary is attempting to access the network through unauthorized pathways. That is, pathways which have not been explicitly allowed by the system administrator.

3.1 Graph-Based Access Control

The goal of this system is to forward information about access control-related events at each service to subsequent services. The accumulated information is used by a policy engine co-located with each service to detect pathway-violation attacks.

At each service, a small program called a *sensor* observes information regarding access-control events and forwards that data to downstream nodes. We packetize this data and call each packet an *event*. Each sensor is situated such that it can observe its target service and report on the access-control decisions made therein. Typically, sensors are quite simple. For example, the sensor for the Apache web server parses the Apache log and error files for reports on authorization attempts. Each entry for an authorization attempt in the log files is an event. The details of the event are associated with it as a set of attribute key-value pairs and reported to downstream services.

Second-order sensors, called correlation sensors, use additional information reported by the sensors to correlate events on a hop-by-hop basis. For example the events generated by the Apache sensor are correlated with packets departing the firewall based on the time, the source port, the IP address, and the TCP sequence number. The complete data set received by a downstream node is a chain, linking the incoming request with the source principal and all intermediate hops in the network. Thus, the policy decision made at a given service can incorporate the additional information obtained from upstream nodes.

As a request propagates through a network, the associated events are forwarded along with it. The events are represented as vertices in a graph, and the correlation information generated by the correlation sensors is used to form edges between them. When a request arrives at a service, it is accompanied by a graph representing the history of its interaction with the network.

Reactive systems like this, as with most intrusion detection systems, depend on the inviolability of the sensor network. This requires particular attention be spent securing the sensors. In this paper, we do not address attacks wherein the sensor network itself is compromised, though we do note that the KeyNote-based system will alleviate some of those attacks. Sensors may be further protected by lifting them into a hypervisory role, or by isolating sensors and applications through virtual machines, as in [23].

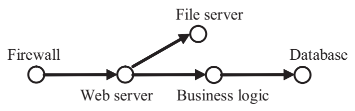

Fig. 3. The tree of applications handling a request.

Note that the overall path taken by a request as it traverses a network is a tree, as in Fig. 3. However, the path taken by a request from its arrival in the network to a given node is a tree-traversal from the root to a leaf or internal node. This path is necessarily linear. As a request passes through a network, the events generated by the sensors associated with it represent the linear path of that request. By situating correlation sensors between hosts and between services, the graph is propagated across the network. Each node in the graph receives the access control decisions made by all its upstream nodes, and this is used to inform future access control decisions.

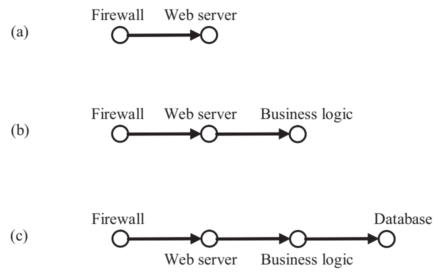

Fig. 4. A graphical representation of the policy at (a) the web server (b) the business logic and (c) the database.

To enact the access control mechanism, we define the policy at each node as a graph, as shown in Fig. 4. The graph representing an incoming connection must match a policy graph in order for the connection to be accepted. However, since the graph is always linear, the policy takes the form of a list of services over which the request must traverse, and valid values for the key-value pairs associated with each event. Any request taking an unexpected pathway or with non-matching attributes will necessarily be detected and rejected.

For example, the policy evaluation at the business logic consists of a traversal of the graph delivered from the upstream node to verify that each node from Fig. 4b appears, and is in the correct order.

3.2 KeyNote-Based Access Control

This method can be viewed as an extension of the graph-based access-control system. In the graph-based system, we build a path representing the route a request took from its entry in the network to a given host. In this system, we leverage the cryptographic tools and trust-management capabilities of the KeyNote system to instead build a certificate chain representing the path taken by a request from its entry in the network to a given host.

Like all reactive, sensor-based systems, the previously-described graph-based system is vulnerable to malicious internal nodes. That is, a compromised or otherwise malicious intermediate node on the path between an application and the entry point for a request can modify the graph dataset before forwarding it. The addition of the KeyNote system protects from some classes of such attacks.

In the KeyNote system, events are reported in the form of KeyNote credentials, and policy is evaluated at each service by a KeyNote compliance checker. Traditional KeyNote credentials allow principals to delegate authorization to other principals, while in the path-based access control scheme, KeyNote credentials delegate authorization for handling a request from a given application to the next downstream application.

When a request generates an event e_1 at host H_1 and the request is then forwarded then to host H_2 where it generates event e_2, a correlation sensor correlates the events as in the graph-based architecture. However, in this case the correlation notification takes the form of a KeyNote credential. That is, it is a signed assertion, with authorizer H_1 and licensee H_2, indicating that e_1 and e_2 are linked. For example, the following credential might be issued by a firewall when it redirects an incoming request to a web server.

```
  KeyNote-Version: 2
  Comment: Forward request to web server
  Local-Constants:  FW_key = "RSA:acdfa1df1011bbac"
                    WEB_key = "RSA:deadbeefcafe001a"
  Authorizer: FW_key
  Licensees: WEB_key
  Signature: "RSA-SHA1:f00f2244"
  Conditions: ...
```

KeyNote provides an additional field `Conditions` which is used to encapsulate references to events e_1 and e_2. Credentials are chained such that the licensee for each event is designated as the next hop in the graph. In a simple e-commerce example, an event generated at a web server and passed to the database would include the previous credential along with the following.

```
KeyNote-Version: 2
  Comment: Send SQL SELECT statement to the DB
  Local-Constants: WEB_key = "RSA:deadbeefcafe001a"
                   DB_key = "RSA:101abbcc22330001"
  Authorizer: WEB_key
  Licensees: DB_key
  Signature: "RSA-SHA1:baba3232"
```

The first link of the credential chain is created by the firewall. This credential binds the principal (the TCP/IP address of the incoming connection) to the first hop in the chain. The key for the principal is randomly generated, and then cached, at the firewall. Such a credential takes the following form:

```
KeyNote-Version: 2
  Comment: New principal at the firewall
  Local-Constants: P_key = "RSA:ffeedd22eecc5555"
                   FW_key = "RSA:acdfa1df1011bbac"
  Authorizer: P_key
  Licensees: FW_key
  Conditions: hop0 == "PRINCIPAL"
  Signature: "RSA-SHA1:ceecd00d"
```

As a request progresses through the network, the result is a chain of credentials that link the incoming request at a given node back through each intermediate node to the principal.

The policy at each node is a list of keys, in order, that must be found in the credential chain. It is similar in concept to the policy definitions shown in Fig. 4, but with each node is also associated a key. As the set of credentials arrives at each node, the local KeyNote compliance checker verifies that the set comprises a chain. If successful, the policy engine then traverses the chain to verify that the keys occur in the order expressed in Fig. 4. If either step fails, the request is blocked.

4 Implementation

Each of these systems were implemented in the Python programming language. Sensors were written and deployed for the OpenBSD PF firewall, Apache, PHP, and MySQL, among other applications. These sensors parse the log files and observe authentication-related behavior of each application in order to generate events describing the access control behavior of each. The correlation sensor engine, an instance of which is deployed between each pair of neighboring sensors, maintains a cache of recently-observed events and generates correlation events based on runtime-configurable fields from the event descriptions. Each time a correlation between two events is made, the two events are linked and forwarded to the next-hop service, along with all previously accumulated events and correlations associated with the request.

At each service, requests are intercepted by a local firewall and redirected to the local policy engine. This engine delays the request until the associated graph arrives from the upstream node. The policy engine traverses the graph and verifies that it conforms to the administrator-defined policy. If the graph validates, the request is allowed to continue to the application, and the graph information is passed to the application sensor.

The KeyNote implementation is similar, but where the graph-based system generated events with arbitrary fields, this implementation generates KeyNote credentials using the KeyNote credential format. The policy for the credential chain is evaluated using the KeyNote compliance checker, through the `pykeynote` module.

5 Evaluation

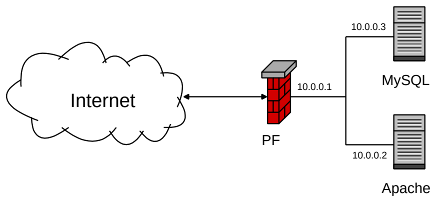

Fig. 5. Testbed network. The OpenBSD PF firewall protects an Apache web server and MySQL database.

We evaluated these two systems on a testbed network consisting of an OpenBSD PF firewall, an Apache web server running PHP 5.2.3, and a MySQL 5.0.45 server. The network is deployed as shown in Fig. 5. The only unblocked incoming port on the firewall is port 80. The firewall also performs network address translation (NAT) so the internal machines have IP addresses in the 10.0.0.0/24 netblock. The testbed application consists of a PHP application which loads and displays a 1MB image from the MySQL database.

The high-level conceptual policy for this network – that is, the policy as it might be expressed informally by the system administrator – is that all connections into this network must be vetted by the firewall to guarantee that they are arriving on the correct port, then processed by the web server and PHP engine, and finally passed to the database. In the attack scenario, a rogue wireless access point is attached to the network as shown in Fig. 6. This opens the potential for incoming connections to access the web server or database without first being processed by the upstream nodes – an *assumption*-violation attack. No events or path information will be generated by requests passing through the wireless access point, since, in this example, the system administrator is unaware of its existence and has not installed any sensors on it.

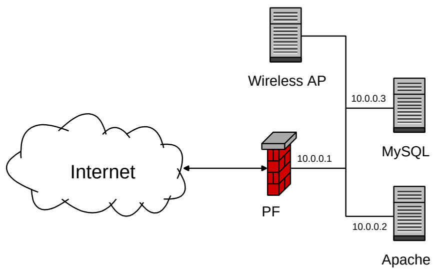

Fig. 6. Vulnerable testbed network. A wireless access point has been connected to the network, allowing traffic to the web server that has not traversed the firewall.

We evaluate the system on two fronts: performance and effectiveness. Performance is measured by timing batch requests made on the system. Effectiveness is analyzed by attempting to detect previously-unseen assumption-violating attacks.

The graph-based access control system is deployed on the testbed network as follows. Sensors are deployed on the network interfaces of all machines, including both network interfaces of the firewall, and at the firewall, web server, PHP engine, and database themselves. Correlation sensors are placed between each neighboring pair of nodes. When a request arrives from an external host, it is processed by the firewall and the sensor on the external network interface. As the request is subsequently processed by the firewall engine itself, and then forwarded out through the internal network interface, the sensors generate events which are linked by correlation sensors. The graph thus generated is collected and forwarded from node to node as the request progresses through the network.

The high-level policy for this network is that all requests must pass, in order, from the firewall to the web server to the database. We derive the actual policy for each node from the high-level policy by determining the path that a request must travel in order to reach that node. Thus, the policy at the database is that it will only handle requests which have traversed the firewall and web server. The policy at the web server is that it will only handle requests that have traversed the firewall. The policy definition for each service consists of an ordered list of nodes. Policy evaluation is a matter of traversing the linear graph built by the sensors and correlation sensors to verify that the nodes occur and are in the correct order.

We test the effectiveness of this system by attempting to connect to the web server and database, through the wireless access point. Since the requests do not pass through the firewall, the graphs associated with the requests, do not have the firewall as the root node. The requests are therefore denied by the policy engine at the web server and database policy engines.

The KeyNote-based system is deployed on the same network. In KeyNote, the policy, rather than being a list of nodes, is a list of keys. The credential chain have have signed credentials, in the correct order, from each of those nodes. *E.g.*, the policy at the database is that the credential chain must have credentials signed by the web server and firewall, in that order. Policy evaluation consists of verification that the credential chain is, in fact a chain, and then a search of that chain for the policy key list.

One test of the effectiveness of the KeyNote system is similar to the tests for the graph-based system. Requests on the firewall are handled as expected, and requests through the wireless access point are blocked as the credential chains thus generated are incorrect.

We analyzing the performance of these systems by determining the overhead incurred by the additional network traffic and processing over the vanilla network. The test application deployed in this network loads files stored in a table in the MySQL database. The test file was 1 megabyte of binary data, and the time for the vanilla system to return that file, from request arrival to completion of the file transfer 162ms, averaged over 25 trials. The average handling time for the graph-based system was 317ms, averaged over 25 trials. The average handling time for the KeyNote-based system was 1.12s, averaged over 25 trials.

The majority of the overhead in the graph-based implementation is due to the delay in waiting for graph information to catch up to each request. The cost increase in the KeyNote-based system is due to the cryptographic costs inherent in the KeyNote system.

We find that in the graph-based system the overhead for a three-node network is 155ms, or approximately 50ms per node. In the KeyNote system, the overhead is 958ms, or approximately 320ms per node. As before, the additional overhead in the KeyNote-based system comes from the substantial cryptographic requirements of the KeyNote architecture.

6 Conclusion

In this work, we have described a mechanism for enhancing the current paradigm of access control to protect against a new class of attacks. These attacks take advantage of the fact that, in the process of converting a security policy from its conceptual, high-level, format to its distributed, low-level, form, information is lost. We describe two systems for defending against this new class of attacks, by passing path information from service to service as the request traverses the network. We show that the overhead incurred in these systems is relatively low.

References

1. Ioannidis, S.: Security policy consistency and distributed evaluation in heterogeneous environments. PhD thesis (2007)
2. Keromytis, A.D., Ioannidis, S., Greenwald, M.B., Smith, J.M.: The STRONGMAN Architecture. In: Proceedings of the 3^{rd} DARPA Information Survivability Conference and Exposition (DISCEX III). (April 2003) 178–188
3. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The KeyNote Trust Management System Version 2. Internet RFC 2704 (September 1999)
4. Blaze, M., Feigenbaum, J., Keromytis, A.: KeyNote: Trust Management for Public-Key Infrastructures. In: Proceedings of the 1998 Cambridge Security Protocols Internatio nal Workshop, Springer, LNCS vol. 1550 (1999) 59–63
5. Damianou, M.: A Policy Framework for Management of Distributed Systems. PhD thesis (2002)
6. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A logical language for expressing authorizations. In: Proceedings of the 1997 IEEE Symposium on Security and Privacy. (May 1997) 31–42
7. Cholvy, L., Cuppens, F.: Analyzing consistency of security policies. In: RSP: 18th IEEE Computer Society Symposium on Research in Security and Privacy. (1997)
8. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., Essiari, A.: Certificate-based access control for widely distributed resources. In: Proceedings of the USENIX Security Symposium. (August 1999) 215–228
9. Keromytis, A.D., Ioannidis, S., Greenwald, M.B., Smith, J.M.: Managing access control in large scale heterogeneous networks. In: Proceedings of the NATO NC3A Symposium on Interoperable Networks for Secure Communications (INSC). (November 2003)
10. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Proc. of the 17th Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos (1996) 164–173
11. Blaze, M., Feigenbaum, J., Strauss, M.: Compliance Checking in the PolicyMaker Trust-Management System. In: Proc. of the Financial Cryptography '98, Lecture Notes = in Computer Science, vol. 1465, Springer, Berlin (1998) 254–274
12. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The role of trust management in distributed systems security. In: Secure Internet Programming. 185–210
13. Ellison, C.: SPKI requirements. Request for Comments 2692, Internet Engineering Task Force (September 1999)
14. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI certificate theory. Request for Comments 2693, Internet Engineering Task Force (September 1999)

15. Ellison, C.M.: SDSI/SPKI BNF. Private Email (July 1997)
16. Bonatti, P., di Vimercati, S.D.C., Samarati, P.: A Modular Approach to Composing Access Policies. In: Proceedings of Computer and Communications Security (CCS) 2000. (November 2000) 164–173
17. Cheswick, W.R., Bellovin, S.M.: Firewalls and Internet Security: Repelling the Wily Hacker. Addison-Wesley (1994)
18. Mogul, J., Rashid, R., Accetta, M.: The Packet Filter: An Efficient Mechanism for User-level Network Code. In: Proceedings of the Eleventh ACM Symposium on Operating Systems Principles. (November 1987) 39–51
19. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: a novel firewall management toolkit. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy. (May 1999) 17–31
20. Hayton, R., Bacon, J., Moody, K.: Access Control in an Open Distributed Environment. In: IEEE Symposium on Security and Privacy. (May 1998)
21. Ioannidis, S., Keromytis, A.D., Bellovin, S.M., Smith, J.M.: Implementing a distributed firewall. In: 7th ACM International Conference on Computer and Communications Security (CCS). (November 2000) 190 – 199
22. Ioannidis, S., Bellovin, S.M., Ioannidis, J., Keromytis, A.D., Anagnostakis, K.G., Smith, J.M.: Virtual private services: Coordinated policy enforcement for distributed applications. International Journal of Network Security (IJNS) **4**(1) (January 2007) 69 – 80
23. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A., Chen, P.M.: Revirt: enabling intrusion analysis through virtual-machine logging and replay. In: OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, New York, NY, USA, ACM (2002) 211–224