CONFIDENTIAL DRAFT: Path-based Access Control for Enterprise Networks

Matthew Burnside, 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 must define policies on a per-application basis, and subsequently, these policies do not interact. For example, there is no mechanism that allows a firewall to communicate decisions based on its ruleset to a web server behind it, even though decisions being made at the firewall may be relevant to decisions made at the web server. In this paper, we describe a path-based access control system which allows applications in a network to pass access-control-related information to neighboring applications, as the applications process requests from outsiders and from each other. This system defends networks against a class of attacks wherein individual applications may make correct access control decisions but the resulting network behavior is incorrect. We demonstrate the system on service-oriented architecture (SOA)-style networks, in two forms, using graph-based policies, and leveraging the KeyNote trust management system.

1 Introduction

Most enterprise networks are distributed structures with multiple administrative domains and heterogeneous components. Defining and enforcing security policies in such networks is challenging. It is difficult for an individual or group of individuals 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.

In an ideal system, an oracle would respond to all security-policy requests from the network. The complete global policy, as defined by the system administrators, could be stored and evaluated at the oracle. However, such an oracle is difficult to construct and clearly does not scale well. Therefore, it is common practice is to derive from the system administrators' high-level conceptual policy a set of policy components where each component is applied to a single application or node. Each policy component is translated into the appropriate language for the target application and deployed directly at that application. In most cases, this task is performed by the system administrator by hand, though there have been some attempts at automating it, as in [22][24].

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 Figure 1. There is a firewall protecting several computers on which are running a web server and some business logic, in the form of, *e.g.*, PHP or ColdFusion, and a database.

Consider a high-level conceptual policy for this network where 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 separate username and password.

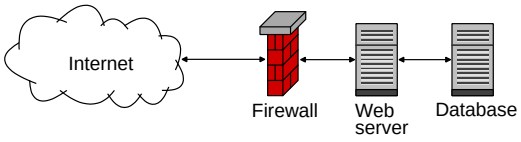

Fig. 1. A simple network. A web server and database are connected to the Internet through a firewall.

The system administrator must take that high-level conceptual policy and from it derive 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 particular username/password may access the files containing the business logic. A final component is the grant table at the database which indicates that only the username/password used by the business logic may access the tables for that application. This is the paradigm under which the policy mechanisms in these applications have been designed, but 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 Figure 2. An adversary can connect to this wireless access point and, through it, connect to the web server and database. Such a connection violates the conceptual policy determined by the system administrator, but not one of the policy mechanisms in place will detect it. That is, none of the policy mechanisms allows for governing how a request arrived at the application, but only what it requests after arrival. Similar network 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 a different path that was supposed to be disabled but was not.

When a global policy is distributed across an enterprise network, as in [21], and each component is evaluated and enforced locally, there is a loss of information. We contend that local evaluation and enforcement of a globally-defined

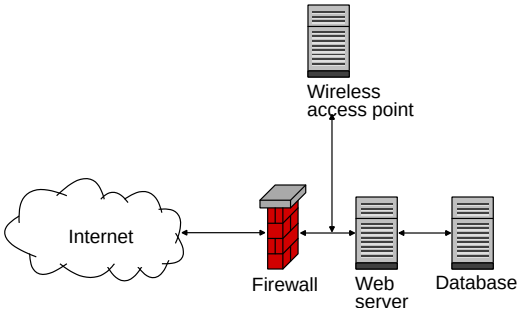

Fig. 2. A vulnerable network.

policy, even when each request is isolated in a virtual service [20], is an incomplete methodology. It fails to address a whole class of attacks, thereby leaving enterprise networks vulnerable to both outsiders and insiders.

Situations where loss of the pathway information may leave an enterprise network vulnerable to an attack are quite common. A compromised internal machine may be used to probe the remainder of the network. A misconfigured router may allow connections to bypass a firewall. A misconfigured firewall may allow connections directly to arbitrary internal machines.

1.2 Contributions

In this work, we attempt to 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 enterprise 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 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 node 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 nodes, it is possible for the policy engine at each node to make decisions based on historical information 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 nodes and dependencies are edges. The second provides protection in some situations where internal nodes are untrusted, and it leverages the KeyNote trust management system[3][4]. We further show that in both cases, the performance overhead is low.

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

2 Related work

In traditional security policy mechanisms, the access-control engine operates as a *gatekeeper* on individual nodes. When a principal makes a request, the access-control mechanism consults a security policy, makes a decision, and goes inactive. The access-control mechanism (and hence, the security policy) is not consulted again, regardless of any future actions taken by that principal. This style of access control was first described by Lampson [26, 27], and refined by Graham and Denning [17]. for specifying security policies in the form of the access control matrix, from which the widely used access control list (ACL) is derived. ACLs consist of a list of tuples:

$$< \text{subject}, \text{object}, \text{access rights} >$$

that define the security policy for the system – which subjects have which access rights on which objects.

ACLs do not scale well in all cases, so in enterprise networks they are often replaced by role-based access control (RBAC) [29, 16, 15]. RBAC is now the predominant model for advanced access control. Each principal is assigned one or more roles, and each role has an associated list of privileges that are permitted members in the role. Both ACLs and RBAC are useful tools, and may play some part in the systems proposed in this paper, but alone they do not solve the problems we have described. In both cases, there paradigm for interaction between individual enforcement points.

Most prior work in the policy field can be divided into three major categories: policy specification [3, 10], resolving policy conflicts [23, 9], and distributed enforcement [30, 25].

In their work in the field of trust management, Blaze, *et al.*, [5, 6, 2] 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 [12–14] is a similar mechanism that uses a formal language for expressing policies. In both cases, the focus is on trust management rather than policy correctness. Both systems could be used as component in a solution to the pathway attacks but alone, they are insufficient.

Bonatti, *et al.*, [7] 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.

When there are multiple policies or multiple users defining policy there is always the possibility of conflict [9]. The problem is exacerbated in large-scale networks.

The STRONGMAN trust management system [24] 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.

Firewalls [8, 28] are one of the most common and most well-known mechanisms for policy enforcement. However, nearly all firewall research has focused on isolated firewall nodes and the specifics of the enforcement mechanisms, rather than policy coordination.

The Oasis architecture [19] 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.

The Firmato system [1] is a firewall management toolkit. 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 the complete policy must be available at the policy-enforcement point so Firmato may not scale well in large networks.

Hale, *et al.*, [18] propose a ticket-based authorization model to manage distributed policies. In this architecture, each network is managed by a controlling mediator communicating with a central policy repository. The mediator serves as a middleware layer, facilitating communication between disparate objects and principals.

Vandenwauver, *et al.*, [31] use a combination of mail, web and script-based attacks to show that any intranet protected solely by firewalls and intrusion detections systems cannot be made completely secure.

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 [20]. 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 our two mechanisms 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 node is a list representing the path from the root to that node 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

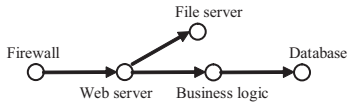

Fig. 3. The tree of applications handling a request.

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

At each application, 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*. Sensors are situated such that they can observe their target applications and report on the access-control decisions made therein. These can often be very 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.

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 based on the time, the source port, and the IP address, with packets departing the firewall. 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 by Apache can be re-evaluated based on the additional information obtained from the firewall's access control decision.

Reactive systems like this, as with most intrusion detection systems, depend on the inviolability of the sensor network. This requires particular attention be spent designing and 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 [11].

Note that the overall path taken by a request as it traverses a network is a tree, as in Figure 3. However, the path taken by a request from its arrival in the network to a given node can be viewed as a tree-traversal from the root to a leaf. 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 applications, 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.

To enact the access control mechanism, we define the policy at each node as a graph, as shown in Figure 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 applications over which the request must traverse. Any request taking an unexpected pathway will necessarily be detected and rejected in this manner.

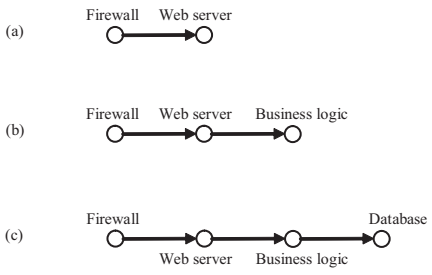

Fig. 4. A graphical representation of the policy at (a) the web server (b) the business logic and (c) the database.

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 Figure 4b appears, and is in the correct order.

3.2 KeyNote-based access control

This system can be viewed as an extension of the graph-based access-control system. In the graph-based system, we build a linear graph representing the path a request took from its entry on the network to a given host. In this system, we leverage the cryptographic tools and trust-management capabilities of the KeyNote system to build a certificate chain representing the path taken by a request from its entry 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 in the network (e.g., login attempts, file requests) are reported in the form of KeyNote credentials, and policy is evaluated locally at each node by the 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 an event is generated at a host A and the request processing then moves on to some second host B, a correlation sensor generates an event in the form of a KeyNote credential. That is, it is a signed assertion describing the form of the event, with authorizer A and licensee B. 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"
```

KeyNote provides an additional field `Conditions`. Here, that field is unused. Credentials are chained such that the licensee for each event is designated as the next hop in the graph. In the e-commerce example, an event generated at the 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 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 then is a list of keys, in order, that must be in the credential chain. It is similar in concept to the policy definitions shown in Figure 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 Figure 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 other 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 and an event describing the correlation between them is forwarded to the next-hop application, along with all previously accumulated events and correlations associated with the request.

At each application, requests are intercepted by a local firewall and redirected to the local policy engine. This engine delays the request in until the 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

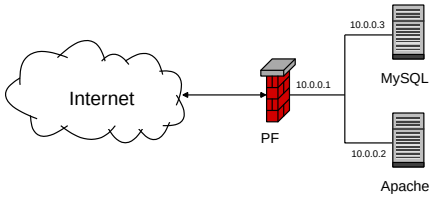

Fig. 5. Testbed network. The OpenBSD PF firewall protects an Apache web server and MySQL database.

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 Figure 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 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 wireless access point is attached to the network as shown in Figure 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.

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

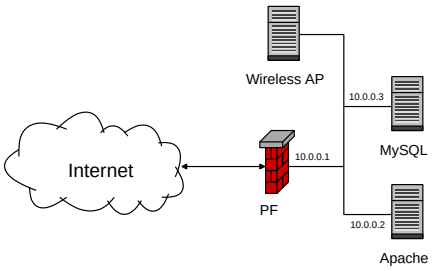

Fig. 6. Vulnerable testbed network. A wireless access point has been connected to the network, allowing traffic to the web server that has not traversed the firewall.

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 conceptual 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 conceptual 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 that 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 application consists of an ordered list of nodes. Policy evaluation is simply 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.

One test of the effectiveness of this system consists of attempting to connect to the web server and database directly, through a wireless access point. As the request does not pass through the firewall, in the case of the web server and the case of the database, the requests are denied.

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 graph-based system was 1.12s, averaged over 25 trials. It is important to note that the overhead is independent of the size of the file being transferred.

The overhead in these systems is due primarily to the intentional delay on each incoming request, until the associated graph information catches up, and then by the policy engine performing a linear-time check on that graph.

Thus, 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. 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 class of access control mechanisms 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, and show that the overhead incurred in these systems is relatively low.

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. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210.
3. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.
4. M. Blaze, J. Feigenbaum, and A. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In *Proceedings of the 1998 Cambridge Security Protocols Internatio nal Workshop*, pages 59–63. Springer, LNCS vol. 1550, 1999.

5. 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.
6. M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in the PolicyMaker Trust-Management System. In *Proc. of the Financial Cryptography '98, Lecture Notes = in Computer Science, vol. 1465*, pages 254–274. Springer, Berlin, 1998.
7. P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. A Modular Approach to Composing Access Policies. In *Proceedings of Computer and Communications Security (CCS) 2000*, pages 164–173, November 2000.
8. W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.
9. L. Cholvy and F. Cuppens. Analyzing consistency of security policies. In *RSP: 18th IEEE Computer Society Symposium on Research in Security and Privacy*, 1997.
10. M. Damianou. *A Policy Framework for Management of Distributed Systems*. PhD thesis, 2002.
11. George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 211–224, New York, NY, USA, 2002. ACM.
12. C. Ellison. SPKI requirements. Request for Comments 2692, Internet Engineering Task Force, September 1999.
13. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. Request for Comments 2693, Internet Engineering Task Force, September 1999.
14. Carl M. Ellison. SDSI/SPKI BNF. Private Email, July 1997.
15. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role Based Access Control*. Artech House, 2003.
16. D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, August 2001.
17. G. S. Graham and P. J. Denning. Protection: Principles and Practices. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 417–429, 1972.
18. J. Hale, P. Galiasso, M. Papa, and S. Shenoi. Security Policy Coordination for Heterogeneous Information Systems. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.
19. R.J. Hayton, J.M. Bacon, and K. Moody. Access Control in an Open Distributed Environment. In *IEEE Symposium on Security and Privacy*, May 1998.
20. Sotiris Ioannidis, Steven M. Bellovin, John Ioannidis, Angelos D. Keromytis, Kostas G. Anagnostakis, and Jonathan M. Smith. Virtual private services: Coordinated policy enforcement for distributed applications. *International Journal of Network Security (IJNS)*, 4(1):69 – 80, January 2007.
21. Sotiris Ioannidis, Angelos D. Keromytis, Steven M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *7th ACM International Conference on Computer and Communications Security (CCS)*, pages 190 – 199, November 2000.
22. Sotoris Ioannidis. *Security policy consistency and distributed evaluation in heterogeneous environments*. PhD thesis, 2007.
23. S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31–42, May 1997.

24. A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith. The STRONGMAN Architecture. In *Proceedings of the* 3^{rd} *DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 178–188, April 2003.
25. Angelos D. Keromytis, Sotiris Ioannidis, Michael B. Greenwald, and Jonathan M. Smith. Managing access control in large scale heterogeneous networks. In *Proceedings of the NATO NC3A Symposium on Interoperable Networks for Secure Communications (INSC)*, November 2003.
26. B.W. Lampson. Protection. In *Proceedings of the* 5^{th} *Princeton Symposium on Information Sciences and Systems*, pages 473–443, March 1971.
27. B.W. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, January 1974.
28. J. Mogul, R. Rashid, and M. Accetta. The Packet Filter: An Efficient Mechanism for User-level Network Code. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 39–51, November 1987.
29. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
30. Mary Thompson, William Johnston, Srilekha Mudumbai, Gary Hoo, Keith Jackson, and Abdelilah Essiari. Certificate-based access control for widely distributed resources. In *Proceedings of the USENIX Security Symposium*, pages 215–228, August 1999.
31. M. Vandenwauver, J. Claessens, W. Moreau, C. Vaduva, and R. Maier. Why enterprises need more than firewalls and intrusion detection systems. In *IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99), 16-18 June 1999, Stanford, CA, USA*, pages p.152–7. Los Alamitos, CA, USA : IEEE Comput. Soc, 1999.