Design and Implementation of Virtual Private Services [*]

Sotiris Ioannidis
sotiris@dsl.cis.upenn.edu
University of Pennsylvania

Steven M. Bellovin
smb@research.att.com
AT&T Labs – Research

John Ioannidis
ji@research.att.com
AT&T Labs – Research

Angelos D. Keromytis
angelos@cs.columbia.edu
Columbia University

Jonathan M. Smith
jms@dsl.cis.upenn.edu
University of Pennsylvania

Abstract

Large scale distributed applications such as electronic commerce and online marketplaces combine network access with multiple storage and computational elements. The distributed responsibility for resource control creates new security and privacy issues, which are exacerbated by the complexity of the operating environment. In order to handle policies at multiple locations, the usual tools available (firewalls and compartmented file storage) get to be used in ways that are clumsy and prone to failure.

We propose a new approach, virtual private services. *Our approach relies on two functional divisions. First, we split policy* specification *and policy* enforcement, *providing local autonomy within the constraints of the global security policy. Second, we create virtual security domains, each with its own security policy. Every domain has an associated set of privileges and permissions restricting it to the resources it needs to use and the services it must perform.* Virtual private services *ensure security and privacy policies are adhered to through coordinated policy enforcement points. We describe our architecture and a prototype implementation, and present a preliminary performance evaluation confirming that our overhead of policy enforcement using is small.*

1 Introduction

Security is an application-dependent property, with some applications requiring very little assistance, while others require considerable infrastructure to support their privacy, integrity and availability requirements. When applications were confined to a single computer, the application programmer could rely on the host operating system to support these requirements. The advent of the Internet has introduced new challenges for applications with non-trivial access-control requirements. In particular, various network access-control mechanisms such as firewalls are largely oblivious to applications (and vice versa), while file-access privileges associated solely with users may not allow for sufficiently fine-grained access control to handle safety issues related to untrusted active content (such as JavaScript applets). In this paper, we introduce the concept of a *virtual private service* (VPS), which captures in a policy specification the complete access-control requirements of a service. This single policy specification can then be used by enforcement mechanisms in hosts, routers, firewalls and elsewhere to produce a consistent environment for the service.

To illustrate virtual private services, let us look at an example. Consider some web services run on a virtual web server consisting of tens or hundreds of machines in a server "farm", with co-located auxiliary services such as a database, credit card transaction support, and web-mail service. Figure 1 shows the components of such a system, without elaborating on the replicas of each component used in a full-scale implementation (*e.g.*,, multiple servers per physical location, and replicated physical locations, each with a database replica and a credit card support system).

Typically, the configuration of such a system is static. By this, we mean that the administrator configures each component independently, and depends on the correctness of the individual policies to enforce a system-wide policy for any particular user or class of users. There is no coordination among the nodes in the system, nor is there any coherent relationship between the network access control (achieved with firewalls and routers) and the node access control. The application components thus become very difficult to manage effectively, and misconfigurations and other adminis-

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

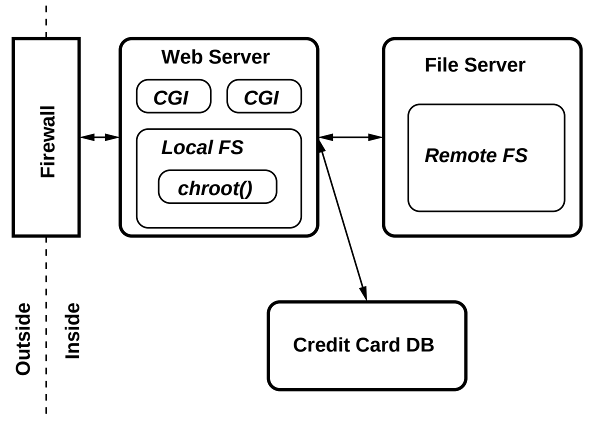

Figure 1. Current general architecture of distributed application clusters.

trative errors creep in [12, 21]. The causes can range from the difficulty of managing local components, the difficulty of managing local scale, or the difficulty of coordinating across sites and administrators. Such systems call for an approach that can ensure consistent access-control policies, as well as meet application-specific requirements using shared resources such as hosts and the network.

Virtual private services are distributed applications which require coordination among clients, servers, and networks to deliver a reliable, secure service to clients. The name is intended to capture the combination of ideas from the virtual private networks used to segregate groups of nodes, and the virtual machine models used to control resources in host operating systems. Our new contribution to this problem is designing and building a system architecture for a *single, ubiquitous security policy* that will be enforced throughout the system (nodes, networks, *etc.*) to meet access-control requirements. Thus, in the web server scenario we sketched earlier, network and host resource access are managed consistently. For example, if an application or user are not permitted to access a service locally, they are prevented from accessing the same service remotely.

2 Motivation

Large distributed systems cannot be administered one machine at a time. This is not, of course, news to system administrators. Many tools (*e.g.,*ASD [15]) have been built to ease the task of administering multiple computers. However, for the most part, these tools have been concerned with *file* management and synchronization/distribution, rather than policies. Policy configuration files can be centrally administered, but this is more a side-effect than a basic premise of the distribution tools. The problem is that complex policies must be expressed in a variety of different ways. For example, consider again the network shown in Figure 1. Assuming there is a security policy barring improper access to the credit card database, the question is how to best architect a system that can enforce this policy.

The first obvious step is a firewall rule that blocks access from the outside. However, we also need to guard against attacks originating from the firewall-protected part of the network, either by insiders or from inside machines that have been compromised. Accordingly, the credit card database may have its own configuration and policy rules blocking most access from “inside” machines. Indeed, it may be protected by its own packet filter or firewall. However, not all users that can legitimately access those “inside” machines should necessarily be trusted. Accordingly, additional access rules may be needed as well. These may be lists of cryptographic credentials to be accepted, or they may be distributed firewall rules [13, 4], *etc.* For that matter, the database system may itself have access control mechanisms that need to be configured.

It is clearly impractical to try to configure each of these systems separately. While current tools can easily distribute policy files, the deeper problem — ensuring consistent access policies, across many different systems — is far more difficult. It is this problem that we are addressing. Our task is further complicated when enforcement must be split across different components. For example, a rule that says “user A may access database column B on server C when coming from machine D via IPsec” should be specified in one place. Enforcement, however, could be split between a firewall that permits access to the database port from D and a firewall rule on D that recognizes A’s credentials, while enforcement of access restrictions to particular fields must be done in the database server itself.

One attempt to solve this problem in a limited domain is the Firmato [3] firewall language. Firmato is a high-level language for specifying firewall policies. The administrator specifies a policy and a network topology; the policy is then compiled into rule-sets for the different firewalls (which may be from different vendors), and distributed to each firewall protecting the domain described in the topology. While this is certainly a step in the right direction — a single policy statement can simultaneously control several different firewalls — it is limited to a single application *class*. As noted above, complex — *i.e.,*realistic — security policies need to simultaneously control many different *types* of applications. Furthermore, the policies must be enforceable *without* the co-operation of the applications, since they may be subverted.

We can thus list our requirements for an effective, multi-layer security mechanism. First, the input language must be rich and extensible, in order to be able to express a wide variety of policies, for a wide variety of devices and applications. Second, the input language must be high-level, to avoid unnecessary device-specific semantics. Third, there must be a reliable compilation and distribution mechanism that will distribute the policy to all relevant network nodes. Finally, the policy must be completely enforceable by trusted components alone (dedicated nodes, operating system kernels, *etc.*), without the cooperation of user-level processes on marginally-trusted machines. However, the definition of a trusted component should be extensible such that a finer-grain policy could be enforced under certain assumptions (*e.g.,*a database that enforces access restrictions to specific columns). Note that proving the enforceability of a given policy in a specific system is a hard theoretical problem; rather than address it, we adopt a pragmatical approach. Our system accomplishes these goals.

3 Architecture

3.1 Separation of Management and Enforcement

The problems we discussed in the previous two sections are exhibited by practically all existing security architectures, and originate from the independent nature of each service. Every application has a different notion of a security policy, performs access control according to that specification, and is oblivious to the security policies of other applications. This often causes configuration problems which lead to security violations.

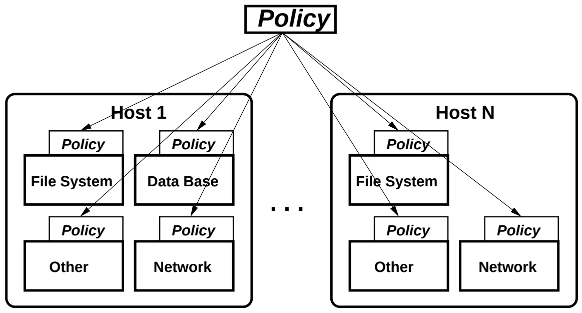

Figure 2. Policy flows from a central specification point to various services. Only the policy rules relevant to a specific service are pulled to that service. No redundant policy state is kept at the access points.

Virtual private services are a new approach to these challenges. Global security policies are specified for services, while enforcement of these policies remains distributed, local to the resource access points. Figure 2 shows how policy is managed in this scheme. The policy flows from a central specification point to the various services. Only the policy rules that are relevant to each specific service are pulled to that service, so no unneeded policy state is maintained at the various access points. In our architecture, we implement policy *management* with the KeyNote [6] trust-management system to express and distribute low-level security policy. Policy *enforcement* is carried out by an augmented host operating system, into which we have inserted several hooks for policy enforcement.

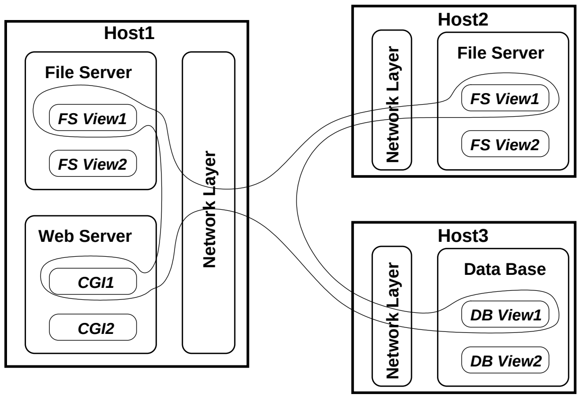

Figure 3. With virtual private services, clients are granted access only to the resources they require to accomplish their task.

Figure 3 demonstrates virtual private services in the context of a web server application. A CGI script running as part of a web server is only given access to specific subtrees of local and remote file systems, a part of a database, and can form network connections only to the machines that host the remote file system and database.

The VPS approach offers several benefits. First, it is *scalable* because policy enforcement is done in a distributed fashion by the access points, avoiding the bottlenecks of a centralized policy server that has to be engaged at policy evaluation/enforcement time. It is *flexible*, since maintenance of policies is centralized and coordinated across different applications. Policy modifications automatically propagate to the enforcement points, *simplifying* the task of administration and management of individual devices and applications. Finally, the VPS approach allows for policy *consistency:* every service added remains consistent with the central security policy. New services cannot diverge from the existing policies.

3.2 Policy Translation and Composition

For our architecture to operate across enforcement boundaries and for policy to be globally enforceable, we include “referral” primitives, first introduced in STRONGMAN [14]; this is simply a reference to a decision made by another enforcement point (typically lower in the proto-

```
Authorizer: ADMINISTRATOR_KEY
Licensees: USER_A
Conditions: ((app_domain == "db access") &&
  (db_column == "column B") &&
  (permissions == "FULL_ACCESS") &&
  (dst_addr == "Server C") &&
  (src_address == "Host D") &&
  (ipsec_result == "YES")) -> "permit";
```

Figure 4. A simplified representation of the VPS policies for the database example from Section 2.

```
Authorizer: ADMINISTRATOR_KEY
Licensees: ANY_USER
Conditions: ((app_domain == "net access") &&
    (src_addr == "ALICE") &&
    (dst_addr == "BOB")) -> "permit";
```

Figure 5. Sample policy for allowing network connections between two machines, from Alice to Bob.

col stack, but more generally at another enforcement point). 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.

To accomplish this, all that is necessary is a channel to propagate this information across enforcement boundaries. In our system, this is done on a case-by-case basis. For example, in our present system 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 to the web server through the SSL data structures.

3.3 Sample Policies

In Section 2 we described an example of a policy for a user accessing a specific column in a database with some additional network constraints. Figure 4 shows how such a policy is described in our system. In this example, the administrator authorizes user A to have full access to the database column B, provided they access it on server C coming from host D over IPsec.

In Section 3.1 we gave a brief example of a service provided by a CGI script (Figure 3). The script requires limited access to the file system (remote and local) and the database, and should not get all the privileges of the web server. We accomplish this by setting up a distributed policy as seen

```
Authorizer: ADMINISTRATOR_KEY
Licensees: ANY_USER
Conditions: ((app_domain == "ftp access") &&
   (directory == "/ftpdir/*") &&
   (permissions == "READ") &&
   (dst_addr == "BOB")) -> "permit";
```

Figure 6. Specification for an FTP policy.

```
Authorizer: ADMINISTRATOR_KEY
Licensees: WEB_ADM
Conditions: ((app_domain == "fs access") &&
 (directory == "/www*") &&
 (permissions == "FULL_ACCESS")) -> "permit";
```

Figure 7. Policy giving the web administrator full access to the WWW directories.

in Figure 9. The first part of the policy guarantees that the script can only connect to either host2 or host3 from host1, the second part will limit file accesses to directories that only contain data for the script, and last part guarantees will only allow the script to access its own database records. The combination of these simple policies assures the properties of the service provided by the CGI script. These subpolicies are independently enforced by the firewall, filesystem, and database server respectively.

Finally, in Figures 5, 6, 7, and 8 we give examples of simple policies that define virtual private services for different users and applications. Administrators can customize services in their system by specifying such policies and guarantee consistency across all system components.

3.4 Evaluation

While the architectural discussion is largely qualitative, some estimates of the system performance are useful. To accomplish this we tested our system with the services for network connection, file system access and web access, defined by the sample policies presented in Section 3.3. Even though the sample services are simple, small scale cases, we believe they provide an adequate picture of the *base* performance of the system. We are currently working on more complex and larger scale scenarios for a more complete evaluation.

In our first experiment we wanted to explore the effects that our architecture has on network performance. For this we set up a simple client to consecutive form TCP connections to a server machine (over 100Mbps Ethernet). The average slowdown due to our access control layer was less than %3 (50.4ms vs. 51.8ms). It took 50.4ms to form the connections on a standard OpenBSD system and 51.8ms

```
Authorizer: ADMINISTRATOR_KEY
Licensees: ANY_USER
Conditions: ((app_domain == "web access") &&
   (directory == "/www/webpages/*") &&
   (permissions == "READ") &&
   (dst_addr == "WEB_SERVER") &&
   (dst_port == "80")) -> "permit";
```

Figure 8. Policy allowing any user to access the web server pages.

```
Authorizer: ADMINISTRATOR_KEY
Licensees: CGI1
Conditions: ((app_domain == "net access") &&
 (src_addr == "Host1") &&
 ((dst_addr == "host2") ||
  (dst_addr == "host3"))) -> "permit";

Authorizer: ADMINISTRATOR_KEY
Licensees: CGI1
Conditions: ((app_domain == "fs access") &&
 (directory == "/www/cgi1data/*") &&
 (permissions == "FULL_ACCESS")) -> "permit";

Authorizer: ADMINISTRATOR_KEY
Licensees: CGI1
Conditions: ((app_domain == "db access") &&
 (records == "cgi1records) &&
 (permissions == "FULL_ACCESS")) -> "permit";
```

Figure 9. Set of polices that apply to the CGI script example from Section 3.1.

when we activated the VPS system.

We then simulated a large file transfer over the network by FTP-ing a 100MB between the server and the client. In this case the our system overhead dropped to less than %0.5 (11,131ms vs. 11,178ms). This reduction is expected, since the cost imposed by our system (invoked once when the network connection is formed and once when the file is first accessed) is amortized over the entire file transfer.

For our final experiment we used `ab(8)`, the Apache web server benchmarking tool. We run it for 500 requests with concurrency 1 and 50, the file transferred was 1024 bytes of static HTML. The resulting overheads were of the order of %1.

4 Related Work

System security for large scale distributed applications is driven by the rapidly changing nature of those applications. The environment we examine in this paper is one of heterogeneous systems, multiple layers of security mechanisms, and great complexity; in that sense, it differs from research focused on single nodes, homogeneous nodes making up a distributed system, or single protocols.

The Flask system [20] extends the idea of capabilities and access control lists with the more general notion of a security policy. Flask relies on a security server for policy decisions and on an object server for enforcement. Every object in the system has an associated security identifier. Requests coming from objects are bound by the permissions associated with their security identifier. Flask does not address the issue of cooperation amongst clients, servers and networks to deliver reliable and secure services to clients.

A different approach relies on the notion of system-call interposition. Systems like Janus [10], Consh [2], and Mapbox [1], operate at user level and confine applications by filtering access to system calls. To accomplish this they rely on *ptrace(2)*, the */proc* file system, and special shared libraries. Another category of systems like Tron [5], SubDomain [8] and others go a step further. They intercept system calls inside the kernel and use policy engines to decide whether to permit the call or not. Our architecture focuses on separation of policy enforcement and specification, and support for distributed compartmentalized services.

Capabilities and access control lists are the most common mechanisms operating systems use for access control. Such mechanisms expand the UNIX security model and are implemented in several popular operating systems, such as Solaris and Windows NT [9]. The Hydra capability based operating system [16] separated its access control mechanisms from the definition of its security policy. Follow up operating system such as KeyKOS [11] and EROS [19] divide a secure system into compartments. Communication between compartments is mediated by a reference monitor. Our system creates distributed compartments using a centralized policy specification.

Traditional firewall work [7] has focused on nodes and enforcement mechanisms rather than overall system protection and policy coordination. There are however proposed firewall architectures [4, 13] that identify the need for flexible policy specification and distribution. Our system reaches beyond networking, extending the set of services that participate in the security domain. Other work that aims to aid the administrator in specifying policies for VPNs can be found in [18, 17].

5 Concluding Remarks

We argued that an increasing number of applications are composed from heterogenous software components interconnected by a network, and that this model introduces new security problems not easily addressed with a conventional set of tools such as compartmented file systems and firewalls. We proposed a new approach, *Virtual Private Services*, which unifies the management of all access-control enforcement points under a single global policy.

Our system copes with scale and heterogeneity, at a low cost in usability, by converting this global policy into a form in which it can be enforced locally. The impact on performance was evaluated using a prototype implementation under the OpenBSD operating system in a series of micro- and macro-benchmarks selected to cover the space of uses of the server side in a distributed application setting. Although our measurements are preliminary, we believe that we have demonstrated that the performance impact of the enforcement mechanism is expected to be low.

In Section 2, we listed four requirements for an effective multi-layer security mechanism. Our system achieves three of the four: we have a rich and extensible language to express policies, a reliable compilation and distribution mechanism of policies to enforcement points, and those policies are completely enforceable. However, we do not feel that our policy language is high-level enough. We intend to investigate the issue of a high-level policy definition language in future research.

The performance analysis ignored the security advantages of virtual private services. We believe that our hypotheses, that is that the cost of the centralized policy specification was low, and that the policy enforcement cost was low, have been demonstrated. Better performance could be gained through recoding and better cache management. Our prototype was only deployed on two hosts as a proof of concept demonstration, we are however interested in deploying the system in a realistic environment. The main goal of this deployment would be investigating the larger-scale (and unfortunately more qualitative) question of the value of a consistent global policy in real systems.

References

[1] A. Acharya and M. Raje. Mapbox: Using parameterized behavior classes to confine applications. In *Proceedings of the USENIX Security Symposium*, pages 1–17, Denver, CO, August 2000.

[2] A. Alexandrov, P. Kmiec, and K. Schauser. Consh: A confined execution environment for internet computations, December 1998.

[3] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *Proc. IEEE Computer Society Symposium on Security and Privacy*, 1999.

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

[5] A. Berman, V. Bourassa, and E. Selberg. TRON: Process-Specific File Protection for the UNIX Operating System. In *Proceedings of the USENIX 1995 Technical Conference*, New Orleans, Louisiana, January 1995.

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

[7] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[8] C. Cowan, S. Beattie, C. Pu, P. Wagle, and V. Gligor. SubDomain: Parsimonious Security for Server Appliances. In *Proceedings of the 14th USENIX System Administration Conference (LISA 2000)*, Mar. 2000.

[9] H. Custer. *Inside Windows NT*. Microsoft Press, 1993.

[10] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A Secure Environment for Untrusted Helper Applications. In *Procedings of the 1996 USENIX Annual Technical Conference*, 1996.

[11] N. Hardy. The KeyKOS architecture. In *Operating Systems Review*, pages 8–25, October 1985.

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

[13] 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.

[14] A. D. Keromytis, S. Ioannidis, M. B. Greenwald, and J. M. Smith. The STRONGMAN Architecture. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 178–188. IEEE Computer Society Press, April 2003.

[15] A. Koenig. Automatic software distribution. In *USENIX Conference Proceedings*, pages 312–322, Salt Lake City, UT, Summer 1984.

[16] R. Levin, E. Cohen, W. Corwin, and W. Wulf. Policy/mechanism separation in hydra. In *Proceedings of the 5th ACM Symposium on Operating Systems Principles*, pages 132–140, November 1975.

[17] I. Lueck, C. Schaefer, and H. Krumm. Model-Based Toll-Assistance for Packet-Filter Design. In *Proceedings of the IEEE Workshop on Policy: Policies for Distributed Systems and Networks*, pages 120–136, 2001.

[18] I. Lueck, S. Voegel, and H. Krumm. Model-based configuration of VPNs. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 589–602, 2002.

[19] J. S. Shapiro, J. M. Smith, and D. J. Farber. EROS: a fast capability system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 170–185, 1999.

[20] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Anderson, and J. Lepreau. The flask security architecture: System support for diverse security policies. In *Proceedings of the USENIX Security Symposium*, pages 123–139, Denver, CO, August 2000.

[21] A. Wool. Architecting the Lumeta Firewall Analyzer. In *Proceedings of the 10th USENIX Security Symposium*, pages 85–97, August 2001.