Policy-Based Management: Bridging the Gap

Susan Hinrichs
Cisco Systems
shinrich@cisco.com

Abstract

In a policy-based system, policy goals are described with respect to network entities (e.g., networks and users) instead of enforcement points (e.g., firewalls and routers). This global view has several advantages: usability, global rules are closer to the goals of the human administrator; scalability, the policy system ensures that the enforcement points are configured appropriately, whether there are 1 or 100 enforcement points; and security, the policy system ensures that the policy is enforced consistently. This paper describes techniques for accurately translating from global policy rules to actual per-device configurations, and it describes how these techniques were used in the implementation of Cisco Secure Policy Manager.

1 Introduction

Policy has been frequently presented as a solution to management problems, but the definitions of policy vary widely. For the purposes of this paper, policy is a global goal statement or constraint. An example of a policy statement is "Engineering should have access to the department web server." This policy statement does not identify the implementation details of which machines belong to engineering and which port the web server is listening on. A policy statement should closely match the goals of the policy decision-maker to reduce the chance of entry error.

For a set of policy statements to be useful, it must be enforced by a set of appropriately configured devices, e.g., firewalls, encrypting-routers, or traffic-shapers. Device configuration is inherently myopic and literal. For example, a firewall does not know which machines are engineering machines. An enforcing firewall needs a rule like "permit TCP traffic on port 80 from 192.168.56.0/24 to 128.45.67.34/32".

There is a conceptual gap between the policy statement and the enforcing configuration that must be bridged to make policy useful in the real world. If there is only one enforcing device involved, the translation is relatively straightforward, but in a larger environment there may be 10's or 100's of enforcing devices that must be coordinated to implement the policy. At this point, the problem of manually doing this translation becomes far more daunting and error prone.

In many ways, this problem is analogous to the problem of compiling a program for a distributed machine[1]. The policy is the program, and the enforcing devices are the nodes in the distributed machine. We can use the same techniques from distributed compilation to perform the translation from policy to a set of consistent device configurations.

This paper describes how we used these compilation techniques in policy-based management during our construction of the Cisco Secure Policy Manager[1][2]. First, we describe the basic components of a policy specification. In Section 3, we describe the steps that a policy compiler performs to translate the global policy to device specific configuration. In Section 4, we provide a concrete example of policy expression and compilation by describing how these features were implemented in the Cisco Secure Policy Manager. We describe evolving policy standards and related policy work in Section 5, and we end with our conclusions in Section 6.

2 Policy expression

A policy statement is a guarded action; when the condition is matched the action constraint is enforced[3]. The policy condition can test against a number of properties. Most commonly, the condition tests against a property of the packet header, e.g., the source IP address or the destination port.

A policy condition can also test against global conditions, e.g., time of day, detected attack, or network

[1] Version 1.0 of this product was called the Cisco Security Manager. Version 2.0 and beyond is named Cisco Secure Policy Manager. For simplicity, this paper refers to all versions as Cisco Security Policy Manager.

load. To make such an external condition useful, the policy-based management system must have access to agents that monitor the state of the world. For example, intrusion detection systems sniff the local network to detect attack signatures and can be used to feed back information about the state of the network into the policy-based management system[4,5].

Finally, a policy condition can test against extended state associated with the network flow, e.g., a user associated with the source IP address. This kind of association also requires some additional infrastructure to be useful. An Authentication, Authorization, and Accounting (AAA) server and a Network Access Server (NAS) can use Radius extensions[6] to change configurations based on an authenticated user.

Policy actions are constraints or requirements associated with the network flows that match the guarding condition. Some policy actions include filtering actions, e.g., permit/deny, block java; cryptographic requirements, e.g., use an encrypting IPSEC tunnel; or quality of service requirements, e.g., give best effort service.

While we have described the policy statements as a set of simple guarded actions, the conditions and actions can be combined into an arbitrarily nested set of conditional statements. Figure 1 shows an example policy describing the constraints on HTTP traffic.

```
If Service is HTTP
  If Destination is S
     If Source is H
        Service level is premium
        Permit
     Else If Source is N1 or N4
        If Source is N4
           Use encrypting tunnel
        Permit
```

Figure 1: Example policy that specifies constraints on HTTP traffic.

Conditional nesting in the policy may aid administrators by allowing them to group features that should be considered together. An arbitrarily nested policy can be flattened into a canonical list form. If the conditional parameters are orthonormal, we can also build optimal search structures to traverse the policy conditions and find the appropriate set of actions[7]. Therefore, deciding whether to nest or to simply require a list of guarded actions is a usability issue not a performance issue.

However, order of the policy rules or policy trees is important. If the user specifies an order of evaluation, the policy-based tool must use this ordering to resolve potential conflicts.

While guarded actions can describe quite complex situations, the policy specification language described above is not Turing complete. There are no looping mechanisms or state assignments. This policy is merely a data flow specification. Without loops, we are guaranteed that evaluating the policy will complete in a fixed amount of time. This guarantee of fixed-time policy evaluation is a must for real-time packet filtering.

2.1 Policy targets

Guarded actions can be used to describe constraints against almost any domain. The action examples above reference the security and quality of service (QOS) domains. In addition, policy has also been proposed for putting constraints on routing. While policy can describe constraints on all these service domains, the operational constraints on these domains differ and these differences can influence the tradeoffs made in implementing a policy-based management system.

The security domain (filtering and cryptography) is the least forgiving of error. If you have a hiccup in the enforcement of your security policy, you may permanently lose connectivity or lose trust by allowing intruders access.

Routing policy has the biggest scaling problem. Huge numbers of routers must be coordinated to consistently enforce the policy goals. For this reason, routing policy tends to be more dynamic and tolerant of changes in the routes.

Depending on your administrative model, QOS policy enforcement falls somewhere between the security domain and the routing domain. An edge-oriented QOS administrative model will be on the same scale as firewall enforcement. A more detailed administrative model will impact more enforcing devices and have more of a scaling concern.

This paper concentrates on the security domain, though the techniques we describe should extend to QOS policy enforcement depending on the QOS administrative model.

3 Policy compilation

Once the administrator specifies policy goals, the enforcing devices must be configured to consistently enforce these policy goals. Traditionally, the administrator or some other technically knowledgeable person has been responsible for creating the device configurations. However, with information about the network topology, this kind of mapping from global intent to local mechanism is well suited to translation automation.

In this section we describe the kind of topology information needed to make this transformation. We also describe the compilation algorithm and various conflict detections and resolutions that can be performed during the translation. We close this section with an example policy translation.

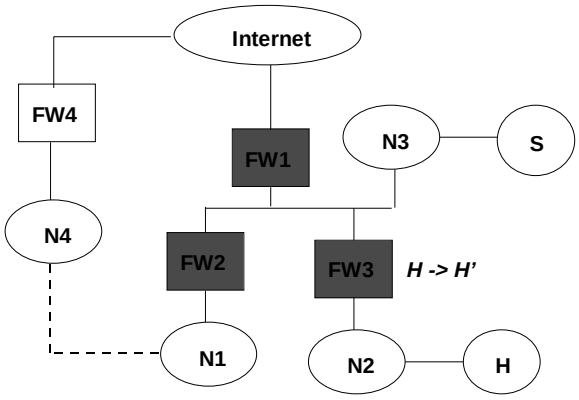

Figure 2: An example topology showing several enforcing firewalls separating networks.

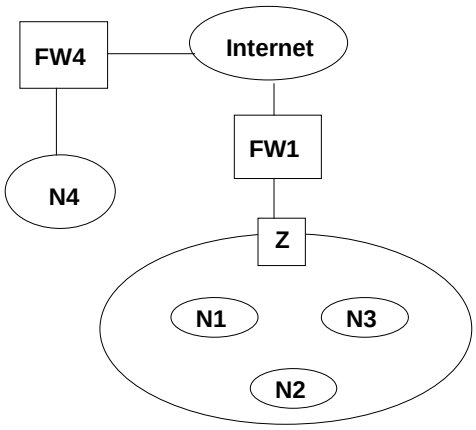

Figure 3: The example topology from Figure 2 with the topology under FW1 collapsed into a single cloud.

3.1 Topology information

The policy compiler must have accurate information about the network topology to perform an accurate mapping from global policy to local configuration. The policy compiler must know the relationships between networks, so it can model the paths where traffic can flow. It must also know the location of all enforcement points under its control.

Figure 2 shows an example topology. If the policy compiler is missing information about a possible path between networks N1 and N4 (as shown by the dashed line), there is the potential for a backdoor because the configuration for the nearby enforcing devices will not enforce the policy between N1 and N4 along that path.

Entering complete topology information by hand is tedious and error prone. Ideally, this topology information can be imported from an already existing database or discovered automatically. Also, in many cases, the complete topology is unnecessary. When implementing a security policy, you only care about the details of the topology near the enforcing devices (firewalls and routers).

The administrator can define a *cloud* to collapse the topology information about a larger set of networks into one virtual gateway that has many networks associated with it. The concept of a network cloud has been used by network designers for years to ignore irrelevant details when discussing a network architecture. The idea is that a subsection of the network can be replaced by a virtual gateway if the details of the network subsection are not relevant to the discussion at hand. Figure 3 shows an example of a network subsection replaced by a network cloud. In this case, we only care that traffic from firewall FW1 must send to default gateway address Z when sending traffic to networks N1, N2, or N3.

3.1.1 Domain of constant policy

The *domain of constant policy* is a concept defined in the Orange Book[8], and this concept appears under one name or another in almost all security management systems, e.g., zones in Lucent Managed firewalls, sites in Centri firewalls. When mapping a policy to a real network, the system must first identify enforcing devices and determine the sets of networks enclosed by the enforcing devices. Each completely enclosed set of networks is a domain of constant policy. By consistently configuring the devices on the perimeter of the domain, a single policy can be consistently enforced against the domain.

The domain of constant policy also defines a lower bound on enforcing granularity. The policy system cannot enforce any constraints on traffic that does not cross the border of a domain of constant policy.

In the topology shown in Figure 2, the gray firewalls enclose a domain of constant policy that includes N3. If firewall FW3 were not there, filtering rules between N2 and N3 could not be enforced.

If an edge of the domain is not protected, there is an unconstrained back door that a knowledgeable opponent could take advantage of. For example, if firewall FW4 were not present and the dashed connection between N1 and N4 were present, we could not reliably enforce any filtering rules about traffic between N1 and the Internet. A rogue program could route along the unprotected dashed link.

While the idea of domains of constant policy has originated in the security world, the same idea is needed in the quality of service arena. Nichols, Jacobson, and Zhang describe a similar idea of policy domains in [9] to automatically negotiate bandwidth requirements between

administrative domains. If the borders are not clearly identified and enforced, rogue traffic could enter and leave the network through an unregulated path.

3.2 Pruning and renaming

Pruning is one of the first steps of compiling a logically shared-memory program to a distributed-memory machine[1]. For each node in the distributed system, the compiler prunes out sections of the program that are irrelevant to that node. In a distributed system with little regularity, you get N pruned programs for N computing nodes. In a distributed system with more topological regularity or a distributed program with more data symmetry, many of the pruned programs are likely to be the same.

Pruning is also the first step in compiling a policy down to the enforcing configurations. The policy compiler steps through each enforcing device and removes all rules that are not relevant to that enforcing device. The policy compiler steps through the global policy rules for each enforcing device. At each source and destination test, the compiler checks whether any path from the source to the destination passes through the target enforcing device. If there is no path through the target device, the test case is pruned out of the rules for the target enforcing device.

As the policy compiler prunes, it also performs address translation rewriting. The topology in Figure 2 shows an address translation rule on firewall FW3. As traffic from host H flows through firewall FW3, its address is changed. The configuration on firewall FW1 must be aware of this address change and enforce rules against the translated address H' rather than the internal address H. The compiler computes the translated addresses at the same time it calculates reachability with respect to the target enforcing device. The algorithm in Figure 4 summarizes the per-device rule calculation.

```
For each policy rule Ri
  devices = CalcPathNodes(Ri.src, Ri.dst)
  For each enforcing device Dj in devices
    Rd = Ri
    Rd.src = SrcAddrWRTDevice(Dj, Ri.src)
    Rd.dst = DstAddrWRTDevice(Dj, Ri.dst)
    Dj.Rules.Append(Rd)
```

Figure 4: The per-device rule generation algorithm.

For a system with R global rules and an average number of D devices on some path for each rule, this pruning calculation will take O(RD) steps. In the worst case the graph is completely connected; D will be the number of enforcing devices in the topology graph.

When calculating the per-device configurations, the policy compiler can also optimize the search structure. Most enforcing devices only take a list of policy rules as input, but if a device can understand nested conditionals, the policy compiler can use d-dimensional trees to optimize the runtime packet search structure[7]. With an ideal d-dimensional tree, the device's runtime rule search should only take O(d + log R) steps where d is the number of conditional elements, i.e., the degree of the search space, and R is the number of rules.

3.3 Consistency checking

The policy compiler can also perform a large number of consistency checks and conflict detection steps.

Is the enforcement point capable of the request? It may be acceptable to have a device incapable of enforcing a request along the path as long as there is another capable device along the path. If it is an expensive check, the user may only want the check performed once even if multiple devices on the path are capable. However, it may be a good idea to leave in multiple security relevant checks particularly if the traffic is moving in and out of controlled networks. A knowledgeable opponent could use spoofing to insert bogus traffic after the only checkpoint on the path.

Does this enforcement point have sufficient resources to carry out the request? On many routers, the fixed set of security associations (SAs) places a strong limit on the number of IPSec tunnels that can be terminated at that router. Memory also places a limit on the number of filtering rules that can be implemented on a given firewall.

Are there conflicts between rules of the same action type? The policy compiler may get two filtering rules that conflict, e.g.,
src=192.168.1.1 dst=10.10.10.10 protocol=IP reject
src=192.168.1.1 dst=10.10.10.10 protocol=IP accept.

If both of these rules come from the same administrative tool, we can rely on the administrator to express an ordering. However, if the rules come from multiple administrative tools, it is not clear which rule takes priority. An external authority must be able to express some sort of priority for each rule set to resolve such conflicts.

Are there conflicts between rules of different action types? Actions that apply to filtering and tunneling may conflict. If the tunnel rules are applied before the filtering rules, the packet header information is no longer available to make filtering decisions. In this case, making a global decision to apply filtering rules before tunneling rules resolves any conflicts between tunneling and filtering.

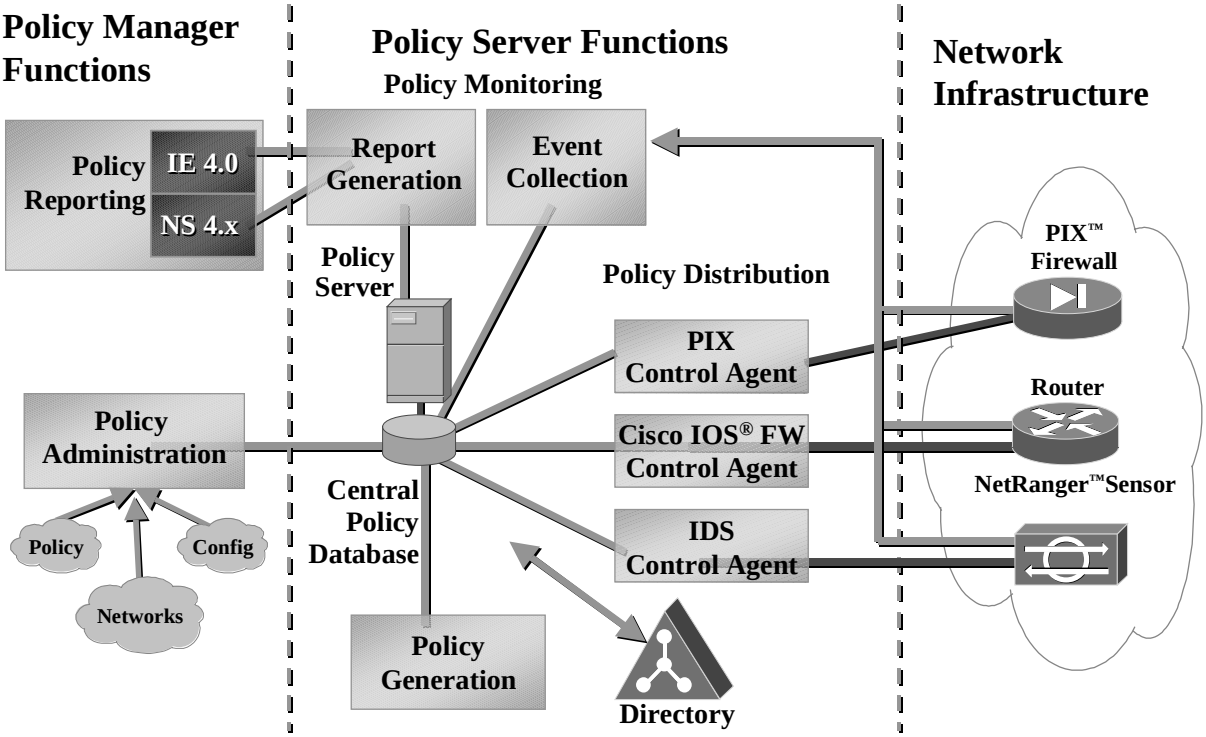

Figure 5: Architectural overview of the Cisco Secure Policy Manager.

Ideally, the policy compiler should be able to detect all conflicts during the initial compilation phase. It is easier and less error prone to centrally orchestrate policy implementation. However, in the real world, the policy compiler must be able to allow for the case where it does not have complete information about the enforcing devices. For example, it may generate a configuration for a device, but the device does not have sufficient memory to load all rules in the configuration.

3.4 Example compilation

Three permit rules arise from the example policy in Figure 1.

1. Svc = HTTP, Src = H, Dst = S, ServiceLevel = premium
2. Svc = HTTP, Src = N1, Dst = S
3. Svc = HTTP, Src = N4, Dst = S, Tunnel = Encrypting

Consider the example topology in Figure 2 without the dashed line connection. Since the topology is a tree, there is exactly one path that each flow can traverse. For rule 1, traffic from H to S traverses enforcing device FW3. In addition to filtering traffic, this device must also be able to enforce the quality of service constraint.

For rule 2, traffic from N1 to S traverses firewall FW2. This rule only specifies a traffic filtering constraint, so a standard firewall should be able to correctly enforce the rule.

For rule 3, traffic from N4 to S traverses enforcing devices FW4 and FW1. In addition to traffic filtering, the rule specifies that the traffic must pass through an encrypting tunnel; therefore, FW1 and FW4 must be capable of being configured as tunnel endpoints.

4 Cisco Secure Policy Manager infrastructure

The previous section described the general theory and issues of mapping global policy to specific device configurations. In this section, we describe how this theory has been implemented in the Cisco Secure Policy Manager architecture.

Over the past three years, our group has worked on a system for mapping user-specified policy to per-device configuration. Our work started with the Centri Firewall[10] and continues with the Cisco Secure Policy Manager[2]. Centri 4.0 controlled a single enforcing device, and it combined the policy expression and topology into a single tree. In Centri 5.0, we had to separate the policy and topology trees to enable policy expression as it applied to multiple enforcing devices. In the Cisco Secure Policy Manager, we expanded the set of target devices beyond the Centri NT-based firewall kernel.

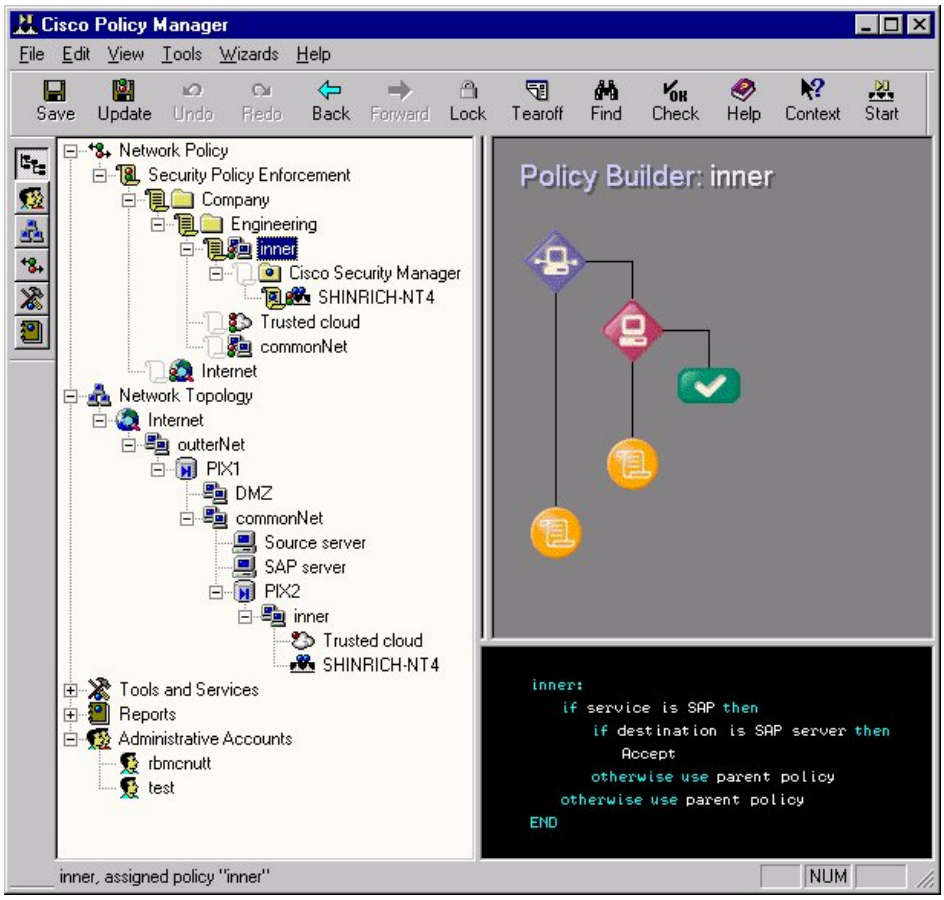

Figure 6: Screen shot of the Cisco Security Manager user interface showing the network topology tree and the policy enforcement tree.

Figure 5 shows an overview of the elements of the Cisco Secure Policy Manager architecture. Version 1.0 compiles policy down to enforcing devices that are PIX Firewalls. Version 2.0 also controls the IPSec features of the PIX Firewall and the firewall feature set and IPSec features of IOS routers. Future versions will also control the NetRanger intrusion detection sensors and other end-to-end network services. The elements of Cisco Secure Policy Manager currently communicate through a proprietary persistent object store

4.1 Administrative interface

The administrator enters policy through a graphical user interface (shown as policy administration in Figure 5). The user interface presents several trees of which two are the most important: the topology tree and the policy enforcement tree. We use these two trees to separate information about the physical relationships from information about logical relationships.

In the topology tree, the administrator specifies the topology, identifies the enforcement points and defines the relations between networks. The topology tree enables the administrator to collapse network details into a network cloud. The current topology editor is restricted to a tree. The topology editor provides a "short-cut" mechanism to express non-tree topologies. A short-cut is much like a soft link in the Unix file system or short-cuts under Windows. With a short-cut, the user can make further references to the same object in multiple places in the topology.

Figure 6 shows a screen shot of the topology tree in the left pane. This example shows a topology with two nested PIX Firewalls. The outer PIX Firewall (PIX1) has a third interface to connect it to a DMZ network in addition to an external interface to the "outterNet" network and an internal interface to the "commonNet". network. Most of the topology

behind the inner PIX Firewall (PIX2) is collapsed into "Trusted cloud."

In the policy enforcement tree, the administrator defines policies using a nested-if specification. These policies are associated with network entity sources. The policy conditions can include information about traffic destination, network protocol, and time-of-day. The policy actions include permit/deny the traffic flow and requirements that the traffic must flow through a particular tunnel. Figure 6 also shows a screen shot of the policy enforcement tree. The policy attached to the inner network is displayed in the right-pane. This policy enables SAP traffic from the inner network to the company SAP server.

The source-based enforcement tree provides structure that enables the administrator to define policy inheritance. The source network objects can be placed in a hierarchy of folders in the enforcement tree. Policies can be attached to the folders or the network objects. The policy evaluation follows a best match algorithm. If we are looking for a policy for a particular host, we first find the object in the enforcement tree that most closely matches, e.g., host, network. Once this is found in the tree, the attached policy is applied. If there is no attached policy, we walk up the parents in the tree until we find a policy. If no policy is found, the default policy to deny all traffic is applied.

When a starting policy is found, it is evaluated with respect to the incoming traffic. If the evaluation ends on a "Use parent policy" node, the evaluation continues on the policy associated with the parent node in the enforcement tree. With this policy inheritance, the administrator can easily describe general policy cases, and then add exceptions for particular machines or users. Figure 7 summarizes this policy inheritance evaluation algorithm.

Policy inheritance makes it easy to make exceptions to a basic policy. In the policy shown in Figure 6, the inner network inherits from the policy attached to the engineering machine policy folder. The policy attached to this folder allows FTP to the departmental source control servers. This policy also inherits from its parent policy attached to the company network folder. This policy allows HTTP traffic to the Internet.

When all elements in the policy enforcement tree are based on an IP address, e.g., network hosts and IP ranges, the meaning of best match is straightforward. However, when authenticated users are allowed as sources of traffic, the meaning of best match is less clear. Suppose Joe has authenticated on the mail server and there is a policy that applies to both entities, which policy takes precedence? In Centri Firewall, user-based policies always took precedence, and in most cases this was what the administrator expects, but in some cases (particularly for server machines) the administrator does not want user-based policies overriding the server's policy. This is an issue that we have not addressed in version 1.0 of Cisco Secure Policy Manager, but it will be an issue when we re-introduce user-based policies in a future version.

```
N = best match node in enforcement tree
for p
while N ≠ ∅
  if N.policy ≠ ∅
    retval = N.policy.Eval(p)
    if retval = accept ∨
       retval = reject
      return retval
    else if retval = Use parent policy
      N = N.parent
  else
    N = N.parent
return reject
```

Figure 7 Summary of the policy enforcement tree evaluation semantics with respect to a particular packet.

After the administrator commits the policy changes, the user interface program stores the proposed policy as a set of global policy objects. These global policy objects are written in the persistent object store.

4.2 Policy compilation

The policy compiler is shown as the "Policy Generation" block in Figure 5. The policy compiler is notified when new policy objects are present in the database. The policy compiler takes the topology information and the global policy objects and generates a per-device policy list in a canonical form as described in Section 3.2. This compiled policy rule list is linked with the enforcing device and stored in the policy database.

The policy evaluation algorithm defined in Figure 7 describes the semantics of the policy enforcement tree evaluation, but it does not describe how the policy must actually be interpreted at run time. The policy compilation phase maps the policy enforcement tree to device-specific configurations. The policy compiler is free to perform semantic preserving transformations in the evaluation tree. In particular, the policy compiler flattens out the inheritance hierarchy and then re-optimizes the common policy rules.

4.3 Policy distribution

A device-specific control agent program is associated with each controlled enforcement point as shown in the "Policy Distribution" block in Figure 5. Multiple control agent programs may be running on different machines in the system to distribute the compilation load and keep the policy control closer to the controlled devices. The control agents perform two main functions: configuration creation and configuration deployment.

4.3.1 Configuration creation

The relevant control agents are notified when new policy rules lists have been generated. The control agent reads the new policy rule list out of the object store and translates the generic policy rule into the syntax of the enforced device. In many cases the translation is a simple syntax transformation from the canonical form to the particular device syntax. In some cases, the transformation is more substantial. For example, the PIX Firewall outbound commands are evaluated based on best match semantics, but the canonical rule set that the policy server generates assumes first match semantics.

For Cisco Secure Policy Manager, the control agents store the per-device configuration into a buffer of commands. When the per-device commands are approved, the control agent telnets in and downloads the commands just as if a human was entering the commands[2]. This is a clunky interface for a program. Seemingly innocuous changes in return messages between device versions can mess up the download program. We chose this mechanism for the first download mechanism because it required no change in the target devices. Future versions will use new download mechanisms better suited for program-controlled download, e.g., Common Open Policy Service (COPS), Lightweight Directory Access Protocol (LDAP).

The system can be run in a mode where the administrator gets a chance to review and approve the translation before it is downloaded to the enforcing device. Until the policy compilation technology matures and gains administrator trust, we expect most administrators to run in this manual approval mode. This is analogous to how programmers would double check compiler output in the early days of compiler technology.

[2] PIX Firewall uses an encrypted secure telnet. IOS can use IPSec to protect the telnet session.

4.3.2 Configuration deployment

Deployment of the approved configurations is a tricky problem that is also encountered in traditional network management. An unwise update order may block a nearer device before its configuration has been updated. For example, consider the topology in Figure 2. Assume machine H hosts the control agent for firewalls FW3 and FW1. If we change the configuration for firewall FW3 before we change the configuration for firewall FW1, we may get into a situation where firewall FW1 is unreachable.

Cisco Secure Policy Manager ensures that basic policy traffic is never prohibited. In addition to making sure that the policy-based traffic is always permitted, the Cisco Secure Policy Manger must ensure that the address translation does not make servers inaccessible[3]. This avoids the problem of being permanently cut off from the control agent.

However, even if the new policy allows the policy download traffic, updating a firewall configuration will likely kill off any existing connections (including the download of the current generation of configurations to another enforcement device).

A complete solution is a two-phase commit. The control agent downloads the new configurations, which the devices store in a separate memory bank. After the control agent detects that all devices under its control have successfully received the new configuration it would give the signal to swap configurations. Unfortunately, most network devices do not have memory to store the next configuration, so this solution cannot be implemented with today's network hardware.

Network management tools generally leave it to the user to specify the download order. The user understands the topology and can determine which devices are farthest away. Cisco Secure Policy Manager has topology information, so it can derive a good download order to help the user.

5 Policy standards and related work

Much standardization work has started in the area of policy-based management. Much of this work has been motivated by quality of service requirements rather than security. However, for such a general problem both policy targets should be able to use

[3] Consider the topology in Figure 2 with address translation rules. Assume the new configuration includes the address translation rule for machine H on firewall FW3. If we download the new configuration for firewall FW3 first, traffic from H will appear to come from H' on firewall FW1, but the original configuration on firewall FW1 only allows configuration traffic from address H. In this case, we must change the configuration on firewall FW1 before changing the configuration on firewall FW3, or we risk permanently breaking connectivity to firewall FW1.

similar techniques. Cisco Secure Policy Manager uses few of these standards because they were not available during development, but we anticipate using the standards as they evolve.

The IETF organized a policy working group to develop policy standards that would apply across multiple target domains. This group is trying to standardize on policy schemas that can be implemented in LDAP directories. With such a standard, policy rules from multiple tools and/or multiple administrative domains can be integrated and made available to standards compliant devices.

To date, this working group has published a draft on core policy schema[3]. The core schema defines the basic guarded agent structure defined in Section 2. All other domain-specific policy schemas should inherit from the core schema.

The COPS protocol has been defined in the RSVP Admission Policy (RAP) working group as a standard protocol for moving policy to the devices. This protocol was developed for the QOS domain, but it is general enough to be used for other target domains. First generation policy-based management tools such as Cisco Secure Policy Manager have relied on telnet and the current command line interfaces. COPS provides a more compact, standard protocol for automating policy changes.

COPS is almost to RFC-status[11] and is implemented in IOS. COPS is a general protocol that can be used in multiple ways. RSVP can use COPS to query policy information from a policy server[12]. COPS can also be used to provision policy information to the device itself for DiffServ[13].

Within the last couple years, the first generation global policy technology has come to light. In [14], Guttman describes a language for describing global filtering policies and algorithms for verifying the policy and generating local filtering rules. Our work differs in the input policy language. Guttman's language assumes that all input police rules are disjoint so policy rule order is unimportant. In our input language, policy rules may conflict so input rule order is important.

More recently Bartal, Mayer, Nissam, and Wool have described their work on the Firmato firewall tool kit[15]. Their work is a similar attempt to derive per-device configurations from a global policy. This work concentrates on firewall filtering. The policy description and inheritance scheme is different from the one described in this paper. The Firmato paper describes a conservative approach to pruning that places all possible enforcing configuration rules on every enforcing device.

6 Conclusions

Policy-based management holds much promise for delivering consistent, correct, and understandable network systems. The benefits of policy-based management will grow as network systems become more complex and offer more services (e.g., security and quality of service).

If the policy system has sufficient information about the network topology, the network administrator can rely on the compiler to take care of the details of generating consistent device configurations. This kind of automated resource management is similar to what existing compilers do, and we can use many algorithms and methods from the compiler world.

We are currently seeing the first generation policy-based management systems. While these first generation systems are useful, there are many areas for improvement in the next few years, e.g. improved download methods, better device support, improved mapping transformations. Policy-based management is proving to be an exciting area promising many rewards to the network administrator.

7 Acknowledgements

Many people at Global Internet and Cisco Systems have contributed over the years to the evolution of the global policy model and compilation algorithms implemented in the current Cisco Secure Policy Manager. These team members include Partha Bhattacharya, Alan M. Carroll, Nick Centanni, Shawn Dempsay, Doug Drew, Brad Frank, Xeuhong Gan, Jimmy Han, Yi Jin, Dima Lebedenko, Imin Lee, Da Li, William Li, Gary Lin, Erica Liu, Blaine McNutt, Tim Petty, Russell Rice, Chis Roulliard, Ken Rowe, Hari Shankar, Liman Wei, Rick Wells, Scott Wiegel, and Bosko Zivaljevic. Thanks also to Robin Lee for doing last minute editing.

8 References

1. D. Callahan and K. Kennedy. *Compiling Programs for Distributed-memory Multiprocessors*. Rice COMP TR88-74, Rice University, August, 1988.
2. Cisco Systems, San Jose, CA. *Cisco Security Manager Tutorial*, DOC-786905, 1999. Available at http://www.cisco.com/warp/public/cc/cisco/mkt/security/csm.
3. J. Strassner, E. Ellesson, and B. Moore. "Policy Framework Core Information Model". Internet Draft, May 17, 1999. Available at http://search.ietf.org/internet-drafts/draft-ietf-policy-core-schema-03.txt.

4. Cisco Systems, San Jose, CA. *NetRanger Overview*, DOC-787019. 1999. Available at http://www.cisco.com/netranger.
5. T. F. Lunt. "A survey of intrusion detection techniques". *Computers and Security* 12(1993):405-418.
6. C. Rigney, A. Rubens, W. Simpson, and S. Willens. "Remote Authentication Dial In User Service (RADIUS)". Request for Comments 2138, Internet Engineering Task Force, April 1997.
7. K. Mehlhorn. *Multi-dimensional Searching and Computational Geometry*. New York:Springer-Verlag, 1984.
8. *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.
9. K. Nichols, V. Jacobson, and L. Zhang. "A Two-bit Differentiated Services Architecture for the Internet", November 1997. Available at http://diffserv.lcs.mit.edu/Drafts/draft-nichols-diff-svc-arch-00.pdf.
10. Cisco Systems, San Jose, CA. *Securing Your Network with Cisco Centri Firewall*, DOC-CENTRIFW-SYN, 1997. Available at http://www.cisco.com/centri.
11. J. Boyle, et.al. "The COPS Protocol". Internet Draft, February 24, 1999. Available at http://search.ietf.org/internet-drafts/draft-ietf-rap-cops-06.txt.
12. J. Boyle, R. et. al. "COPS usage for RSVP". Internet Draft, February 26, 1999. Available at http://search.ietf.org/internet-drafts/draft-ietf-rap-cops-rsvp-04.txt.
13. F. Reichmeyer, et.al. "COPS Usage for Policy Provisioning". Internet Draft, February 1999. Available at http://search.ietf.org/internet-drafts/draft-sgai-cops-provisioning-00.txt.
14. J. D. Guttman. 1997. "Filtering Postu