Firmato: A Novel Firewall Management Toolkit

YAIR BARTAL
The Hebrew University of Jerusalem
ALAIN MAYER
CenterRun Inc.
KOBBI NISSIM
Microsoft Research, SVC
and
AVISHAI WOOL
Tel Aviv University

In recent years packet-filtering firewalls have seen some impressive technological advances (e.g., stateful inspection, transparency, performance, etc.) and wide-spread deployment. In contrast, firewall and security *management* technology is lacking. In this paper we present *Firmato*, a firewall management toolkit, with the following distinguishing properties and components: (1) an entity-relationship model containing, in a unified form, global knowledge of the security policy and of the network topology; (2) a model definition language, which we use as an interface to define an instance of the entity-relationship model; (3) a model compiler, translating the global knowledge of the model into firewall-specific configuration files; and (4) a graphical firewall rule illustrator.

We implemented a prototype of our toolkit to work with several commercially available firewall products. This prototype was used to control an operational firewall for several months. We believe that our approach is an important step toward streamlining the process of configuring and managing firewalls, especially in complex, multi-firewall installations.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection (e.g., firewalls)*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

A preliminary version of this article appeared in the *20th IEEE Symposium on Security and Privacy*, Oakland, CA, May 1999. Work done while all the authors were with Bell Labs, Lucent.
Authors' addresses: Y. Bartal, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel; email: yair@cs.huji.ac.il; A. Mayer, CenterRun Inc., 900 Island Drive, Redwood City, CA 94065; email: alain_mayer@hotmail.com; K. Nissim, Microsoft Research, SVC., 1065 La Avenida, Mountain View, CA 94043; email: kobbi@microsoft.com; A. Wool, School of Electrical Engineering, Tel Aviv University, Ramat Aviv 69978, Israel; email: yash@acm.org.

General Terms: Management, Security, Languages

Additional Key Words and Phrases: Security policy, model definition language, visualization, firewall management

1. INTRODUCTION

Firewalls are a fact of life for companies that are connected to the Internet. However, firewalls are not simple appliances that can be activated "out of the box." Once a company acquires a firewall to protect its intranet, a security/systems administrator has to configure and manage the firewall to realize an appropriate security policy for the particular needs of the company. This is a crucial task; quoting from Rubin et al. [1997]: "The single most important factor of your firewall's security is how *you* configure it." However, while firewalls themselves have seen some impressive technological advances (e.g., stateful inspection, transparency, performance, etc.), firewall configuration and management seem to be lagging behind. Indeed, a Forrester report shows that the clear winner on Fortune 1000 companies' wish lists for Internet security enhancement is "security management tools" [Howe et al. 1996].

Before we describe our approach, we briefly examine what makes firewall management a difficult task. A firewall is typically placed on a gateway, separating the corporate intranet from the public Internet. Most of today's firewalls are configured via a *rule-base*. In the case of a firewall guarding a single, homogeneous intranet (e.g., a small company LAN), a single rule-base instructs the firewall which inbound sessions (packets) to let pass and which to block. Similarly, the rule-base specifies what outbound sessions are allowed. The administrator needs to implement the high-level corporate security policy using this low-level rule-base.

The configuration interface typically allows the security administrator to define various host-groups (ranges of IP addresses) and service-groups (groups of protocols and corresponding port-numbers at the hosts, which form the endpoints).[1] A single rule typically includes a *source*, a *destination*, a *service-group*, and an appropriate *action*. The source and destination are host-groups, and the action is either "pass" or "drop" (the packets of the corresponding session).[2] In most firewalls, the rule-base is order-sensitive: the firewall checks if the first rule in the rule-base applies to a new session. If so, the packets are either dropped or let through according to the action of the first rule. Otherwise, the firewall checks if the second rule applies, and so forth. When we examined real-life firewall rule-bases, we saw that this scheme often leads to mis-configuration in which there are redundant rules in the rule-base, and the desired security policy is realized only after re-ordering some of the rules. Another possible mistake is to set up the rules so that the firewall gateway is completely unreachable,

[1]This is the case for Check Point FireWall-1 [Welch-Abernathy 2002] and many other firewall products. Notable exceptions are Cisco products, like the Cisco PIX Firewall [Chapman and Fox 2001]. The Cisco PIX Firewall does not support service groups at all, and only allows host-groups that consist of a single subnet, defined via an IP address and a net-mask.

[2]Other actions are usually allowed, such as writing a log record or forwarding the packets. We focus only on the basic pass/drop actions, for sake of brevity.

and it becomes impossible to download new rule-bases. The bottom line, however, is that the security of the whole intranet depends on the exact content of the rule-base, with no higher level of abstraction available. Since the syntax and semantics of the rules and their ordering depend on the firewall product/vendor, this is akin to the dark ages of software, where programs were written in assembly language so the programmer had to know all the idiosyncrasies of the target processor.

These problems get even worse for medium- or large-sized companies that use more than a single firewall, or use routers for internal packet filtering tasks. These filtering devices divide the company's intranets into multiple *zones*, such as human resources, research, and so forth. In this case, the security policy is typically realized by *multiple* rule-bases, located on multiple gateways that connect the different zones to each other. Thus, the interplay between these bases must be carefully examined so as not to introduce security holes. A quote from the same Forrester report confirms that "security managers need a single place to look for the corporate policies on who gets in and who doesn't." It is easy to see how rapidly the complexity of designing and managing these rules grows, as intranets get more complex. There is strong evidence that corporate firewalls are often mis-configured [Wool 2004b].

Our objective is to design and implement a firewall management toolkit that has the following distinguishing properties:

(1) *Separate the security policy design from the firewall / router vendor specifics*. This allows a security administrator to focus on designing an appropriate policy without worrying about firewall rule complexity, rule ordering, and other low-level configuration issues. It also enables a unified management of network components from different vendors and a much easier transition when a company switches vendors.
(2) *Separate the security policy design from the actual network topology*. This enables the administrator to maintain a consistent policy in the face of intranet topology changes. Furthermore, this modularization also allows the administrator reuse the same policy at multiple corporate sites with different network details, or to allow smaller companies to use default/exemplary policies designed by experts.
(3) *Generate the firewall configuration files automatically* from the security policy simultaneously for multiple gateways. This reduces the probability of security holes introduced by hard-to-detect errors in firewall-specific configuration files.
(4) *Allow high-level debugging of configuration files*. This allows a security administrator to capture and reason about the information in the configuration files.

1.1 Our Results

We have designed and implemented the following components of the *Firmato* firewall management toolkit, as illustrated in Figure 1:

(1) *An Entity-Relationship Model*, which provides a framework for representing both the (firewall independent) security policy and the network topology.

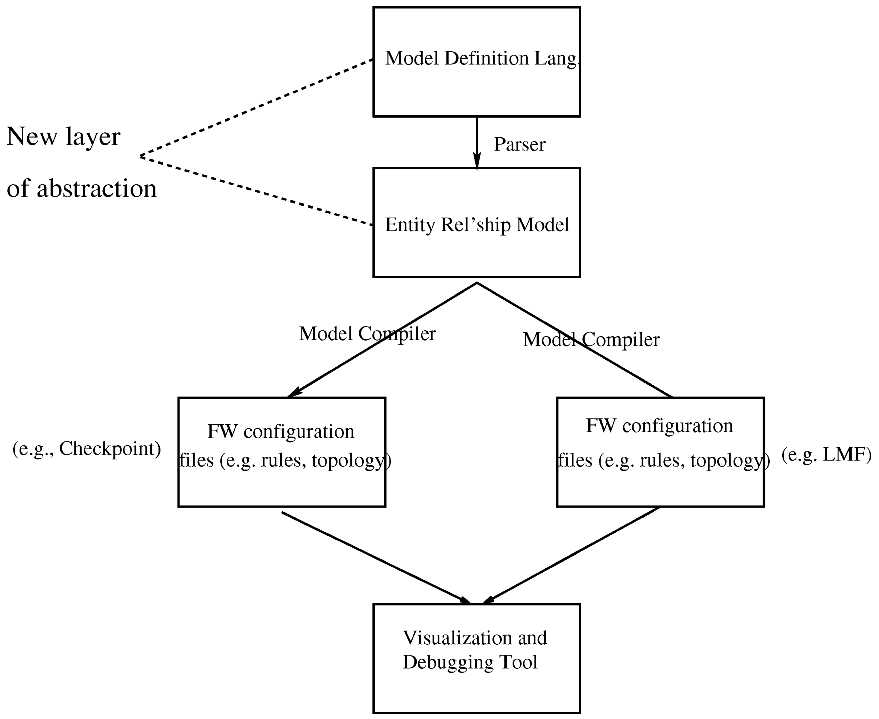

Fig. 1. Toolkit components.

We achieve this by expressing the security policy in terms of *roles*, which we use to define network permissions. Roles capture the topology- and firewall-independent essence of a policy.

(2) *A Model Definition Language* (MDL), which we use as an interface to define an instance of the entity-relationship model. We have implemented a parser for MDL that generates such instances.

(3) *A Model Compiler*, which translates a model instance into firewall/router-specific configuration files. A set of such files typically includes topology and rule-base information. Rules typically need to be distributed onto several filtering devices, in which case the compiler has to generate a separate set of local configuration files for each device. The compiler can thus be further sub-divided into the generation of a central base of logical rules, the distribution of these rules onto various devices, and the transformation of a logical rule into one or more vendor-specific rules or access control lists.

(4) *A Rule Illustrator*, which transforms the firewall-specific configuration files into a graphical representation of the current policy on the actual topology. Such a visualization allows a quick first evaluation of the viability of a chosen policy.

An early implementation of *Firmato*, which we shall call *Firmato-v1*, was described in Bartal et al. [1999]. In this article we describe the current state of

Firmato, which we refer to as *Firmato-v2* when we wish to distinguish its new features.

1.2 Related Work

There are many firewall products on the market, from vendors such as Check Point [Welch-Abernathy 2002], Cisco [Chapman and Fox 2001; Held and Hundley 1999], Lucent Technologies [Lucent 2002], Symantec, and Network Associates, just to mention a few (see ICSA Labs [2003] for an updated list of vendors). Additionally, there are many books on firewall technology (e.g., Cheswick et al. [2003]; Chapman and Zwicky [1995]; Welch-Abernathy [2002]; Chapman and Fox [2001]). While most of the firewall offerings include configuration tools with varying degrees of sophistication, their focus does not seem to be on firewall and security *management* tools. This tendency is slowly changing, though, and we are currently seeing a few first-generation products being introduced in this arena, for example, by Check Point, Cisco [Hinrichs 1999], and Solsoft [Solsoft 2000].

The research work closest in spirit to ours is probably Guttman's work on filtering postures [Guttman 1997; Guttman 2001]. There, a Lisp-like definition language is introduced to define a filtering policy. Also, a method for localizing the policy to the different interfaces of a filtering router is given (where the local policies are again expressed in the same Lisp-like formalism). A similar approach was also used in Guttman et al. [2000] to analyze IPsec policies. While an important step toward security management, Guttman [1997] does not provide complete separation of the security policy from the network topology or automatic generation of firewall rules. The first issue also makes policy modularization and reuse much harder. Furthermore, while the specification is firewall independent, it (and the localization method) does not seem to easily lend itself to be used with actual firewalls.

Open-source tools that support multiple flavors of packet filters can be found, for example, in Reed [2002]; HLFL [2002]. These tools provide a uniform configuration language for the platforms they support, however they do not model the topology (so they do not configure multiple devices from the same policy) and their input language has roughly the same level of abstraction as the underlying filter languages.

Recently there has been a renewed interest in firewall research, focusing on Bellovin's idea of a distributed firewall [Bellovin 1999]. A working prototype has been developed under OpenBSD [Ioannidis et al. 2000]. The basic idea is to make every host into a firewall that filters traffic to and from itself. The main advantages of a distributed firewall are that (i) since the filtering is at the endpoint, it can be based on more detailed information (such as the binary executable that is sending or receiving the packets); and (ii) there is no bandwidth bottleneck at the perimeter firewall. The main difficulties with a distributed firewall are (i) the need for a central policy to control the filtering, and (ii) the need to ensure that *every* device in the network is protected, including infrastructure devices like routers and printers. Therefore, we believe that the advent of distributed firewalls will only increase the need for more sophisticated policy management tools.

Role-based access control (RBAC) (see Sandhu [1998] for a survey) and research in trusted/secure operating systems with adaptive security policies (see, e.g., Carney and Loe [1998]) have a somewhat similar flavor to our work in the sense that they also strive to separate the security policy from the underlying enforcement mechanism. However, their focus is mostly on assigning permissions to (human) users of computing systems, whereas we deal with assigning permissions to IP addresses in the context of a network topology with different enforcement devices. Nevertheless, with additional foundational work, RBAC may possibly be extended to apply to network security. The notion of *roles* was also introduced for authentication in distributed systems in Lampson et al. [1992]; there a principal can act in a role to express acting as a different, weaker principal.

Organization. In Section 2 we introduce our entity relationship model in some detail. In Section 3 we describe the syntax of the model definition language MDL. Section 4 covers the design of our topology-independent rule compiler, and Section 5 covers the rule-to-interface distribution algorithms and the device-specific back-ends. In Section 6 we discuss the rule illustrator. We then walk the reader through a complete realistic example of using our toolkit in Section 7. Section 8 covers *Firmato*'s field deployment. We describe the lessons we have learned and mention some future work in Section 9.

2. THE MODELING FRAMEWORK

2.1 Terminology

Firewall terminology varies slightly from vendor to vendor, so we need to precisely define the terms we use.

Gateways. These are the packet filtering devices that are to be configured. Gateways can be firewalls, routers, bridges, or other devices that filter IP traffic.

Interfaces. Typically, a gateway has multiple network connections. Each connection goes through an *interface*. We assume that each interface has a packet filtering rule-base associated with it: this is more general than assuming only a single rule-base per gateway.[3] Normally each interface has its own unique IP address. However, this is not the case when the gateway operates as a layer-2 bridge [Limoncelli 1999].

Zones. The gateways partition the IP address space into disjoint *zones*. Precisely, a zone z is a maximal set of IP addresses such that packets may be sent between any two addresses in z without passing through any gateway. Most zones correspond to a corporation's subnet(s), usually with one big "Internet" zone corresponding to the portion of the IP address space that is not used by the corporation. Note that zones are required to be disjoint: each IP address can only appear in a single zone.

[3]We assume that a rule includes a direction field that specifies whether traffic is entering or exiting the gateway. This is equivalent to assuming that each interface has 2 rule-bases associated with it, one for each direction.

Service. This is the combination of a protocol-base (e.g., `tcp`, `udp`, etc.) and the port numbers on both the source and destination sides. For instance, the service `telnet` is defined as `tcp` with destination port 23 and any source port. For the `icmp` protocol, we specify the message type and code instead of port numbers.

Remark. The zones are defined by those gateways that are actually *modeled*: the gateways that have the ability to filter packets, and are configured by the same administrator. Typically these gateways only include the perimeter firewalls and the internal gateways that function as "bulkheads" between parts of the internal network: there is no need to model every router in the system.

2.2 Modeling Concepts

2.2.1 *Modeling Phases.* In every organization's network there exist different collections of machines that will be assigned different functions. The heart of a security model is to decide which of these collections should engage in which services with which other collections. A central theme in our modeling framework is that it is valuable to express these decisions *without saying what hosts or devices belong to which collections, or where any of these hosts or devices are located.* With slight abuse of standard terminology, we call all the information about IP addresses, subnets, and the way they are connected, the *topology*.

Therefore, in our modeling framework, the network administrator goes through two separate phases in developing a model. In the first phase, the administrator selects symbolic names, which we call *roles*, and describes which services are going to be permitted between whatever machines that are assigned this role and whatever machines that are assigned peer roles. This is what we call the topology-independent part of the security policy. In the second phase, the administrator specifies the assignments of roles to actual hosts.

2.2.2 *Roles, Role Peers, and Permissions.* The main concept we introduce is that of a *role*. A role is a name that will be assigned to a set of hosts. It summarizes hosts that will share a functional attribute. All hosts given the same role will have the same permission to initiate or accept a certain set of services. For example, the role of a `mail_server` might define the permission of accepting mail service (`smtp`, i.e., `tcp` at port 25) from anywhere. For convenience we also allow *role-groups*, which are names for unions of roles, that can be assigned together to hosts or host groups.

Note that a packet is always traveling between two hosts: from the source to the destination. Therefore, in our framework, a permission to establish a connection using some service is actually a property of *two* roles, which we refer to as *peers*. Defining a role of `mail_server` and assigning it to some host h will not, on its own, let clients connect to to h and send mail. A peer role, such as `mail_client`, has to be defined, and the `smtp` service has to be permitted from the the `mail_client` role to its `mail_server` peer role. If we would like hosts with role `mail_server` to accept `smtp` connections from anywhere, we need to permit `smtp` between the peers "*" (a pre-defined role group, which means "all roles") and `mail_server`.

Roles control *positive* permissions, and implicitly realize a "whatever is not explicitly allowed is disallowed" strategy, for example, a host accepts an `http`-request if and only if it was assigned a corresponding role of `web-server`. Using positive statements has well-known formal-logic benefits: The logic is always consistent, and the statements commute. This last property means that the firewall policy is indifferent to the order of rules that are generated—which simplifies our rule generation algorithms.

Remark. The mapping between our concepts and those used in Role-Based Access Control (RBAC) is: RBAC users map to hosts, and RBAC permissions map to services. However, unlike RBAC, assigning a role to a host is not enough in our model: A permission is always given to a pair of source/destination hosts, each of which has been assigned the appropriate role.

2.2.3 *Hosts, Host-Groups, and Inheritance.* A *host* entity models a machine with an IP address. A *host-group* represents a named set of machines, defined as an arbitrary collection of IP addresses. The basic object we use to define collections of IP addresses is a range of IP addresses. So, precisely, a host-group consists of an arbitrary collection of IP address ranges, and a host is an IP address range which consists of a single address.

We allow a host-group to be defined explicitly by its IP addresses, or via set union and set difference operations between already-defined host-groups. Roles can be attached to both host and host-group entities.

Host-groups have the usual set containment and set intersection relations defined by the sets of IP addresses they consist of. These relations naturally imply an *inheritance* of roles: The set of roles assumed by a host h is the union of all the roles assigned to host-groups that contain h.

Remarks.

—A zone (recall Section 2.1) is just a host-group with additional attributes that describe its relationship with the adjacent getaways, and with the property that zones are disjoint.

—A host-group is not required to be contained in one zone. Defining zone-spanning host-groups is often natural, and such definitions are very common in actual firewalls. Unfortunately, they are also the cause for various firewall misconfigurations [Wool 2004b]. *Firmato* allows such definitions, despite the problems they create. See [Wool 2004a] for a more in-depth discussion.

2.2.4 *Negative Expressiveness.* Recall that tuples of (source role, destination role, service) express *positive* permissions. Note that such role/role/service tuples (and thus positive firewall rules alone) have enough expressive power to realize any policy. This can be achieved, albeit somewhat tediously, by refining the host-groups so that they become disjoint—then no inheritance of roles can occur. However, negative statements are often natural to users, and sometimes allow a more efficient implementation of the policy (using fewer rules).

Consider, for instance, a firewall with interfaces I_1 and I_2, connected to zones Z_1 and Z_2 respectively. It is important to ensure that the firewall itself is "stealthed": access to it should be limited to a few administrative tasks from a small set of other machines, and it should be invisible to all other machines. The difficulty is that the IP address of each of the firewall's interfaces belongs to its adjacent zone. While it is easy to assign very restrictive roles to the firewall's interfaces, the firewall might inherit other roles, which are assigned to the "regular" hosts in the adjacent zones. To avoid such side-effects using positive permissions only, the administrator would be forced to define an artificial host-group consisting of (the IP addresses in) "$Z_1 \setminus I_1$."

The design issue thus becomes how to allow a simple, yet adequate, form of negative expressiveness to the security administrator. For this purpose we introduce the notion of a *CLOSED* group of roles. A *CLOSED* role-group is a role-group for which inheritance of roles does not apply: a host h that is assigned a *CLOSED* role-group does *not* inherit other roles assigned to any host-group A that contains h. A host may be assigned at most one *CLOSED* role-group. Role-groups that are not *CLOSED* are called *OPEN*. By default role-groups are *OPEN*.

By controlling role inheritance, the "*CLOSED*" mechanism allows a limited form of negative expressiveness and allows *group exclusion*. This is reminiscent of the approach taken by Swift et al. [2002], in the design of the Windows 2000 access control lists.

Continuing the above example, the administrator could avoid the artificial host-group definition "$Z_1 \setminus I_1$" by associating a *CLOSED* role group with I_1. Since I_1 assumes a *CLOSED* role-group, it would not inherit roles assigned to Z_1—even though I_1's IP address belongs to the zone Z_1.

When we designed *Firmato*, we believed that the "*CLOSED*" mechanism captures just enough negative expressiveness. However, with hindsight, this mechanism may have been too limiting. See Section 9.3 for a discussion of this issue.

2.3 Our Entity-Relationship Model

We now describe our model framework in some more detail. See Figure 2 for a pictorial representation; we note that for ease of representation, we omit some of the attributes and objects from this discussion.

A `Role` entity consists of a set of `Peer-Permissions`. Each such `Permission` defines (via its attributes) the allowed `Services`, the `Peers`, and the `Direction` in which the service is allowed to be executed (i.e., from the role to the peer for an outgoing permission, or from the peer to the role for an incoming one). The `Service` entity has a `Protocol-Base` and two port number attributes `Dest-Port-No-Range` and `Src-Port-No-Range`. A `Peer` points to another (or the same) `Role`.

A `Role-group` entity consists of a set of `Roles`. It also has a Boolean attribute `Closed`, which designates the role-group as a *CLOSED* one. Recall that hosts that are assigned a *CLOSED* role-group do not inherit other roles.

We model the network topology as follows: the network is partitioned into `Zones`, connected through `Gateways`. A `Gateway` consists of a `Gateway-Interface` for each adjacent `Zone`. A `Gateway-Interface` usually has its own IP-address (modeled by the `Interface-Host` attribute). However, we also allow `Invisible`

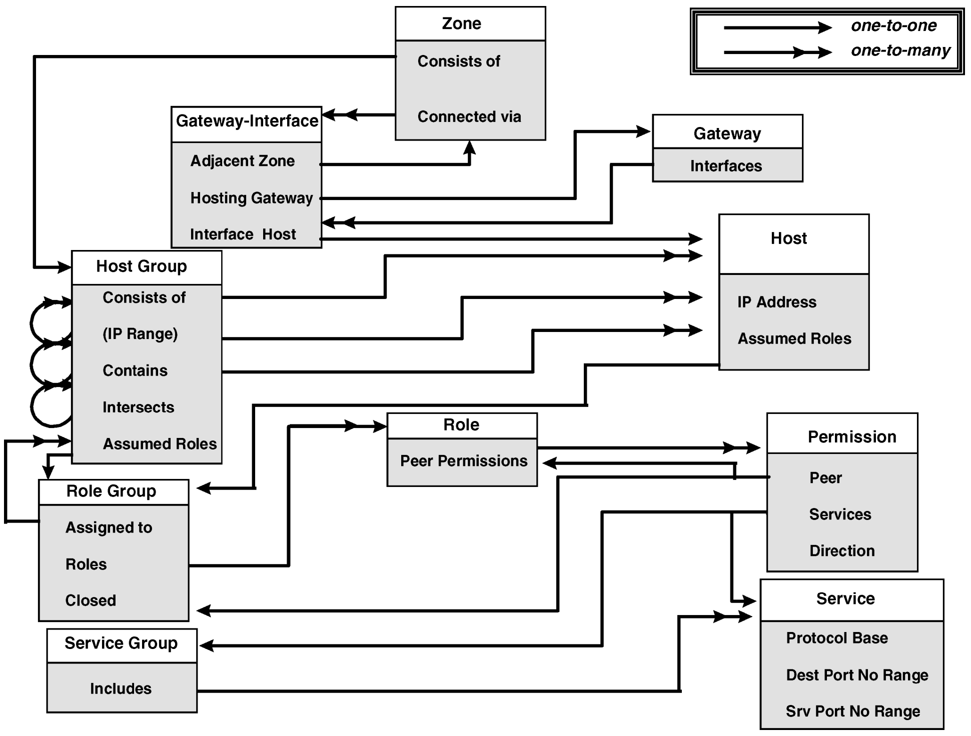

Fig. 2. The entity-relationship model.

interfaces, to model bridge firewalls operating in layer 2. Packets leaving and entering a `Zone` can be filtered by the gateway on the corresponding `Gateway-Interface`; packets sent and received within the same `Zone` cannot, simply because they do not pass through any gateway. `Zones` consist of host-groups (`HostGrps`). `HostGrps` are typically further subdivided into a hierarchy of smaller `HostGrps` or single `Hosts`. Each `HostGrp` stores its containment and intersection relationship to other host-groups in its `Contains` and `Intersects` attributes.

`HostGrps` and `Hosts` are the entities to which we attach role-groups (via the attribute `AssumedRoles`) therefore this is the place where the policy (modeled by roles and role-groups) is linked to the network topology.

3. THE MODEL DEFINITION LANGUAGE MDL

We have developed a simple model definition language MDL as a method of instantiating a security policy, and the mapping of the policy onto the topology. Here we give a definition-by-example of MDL; an example with a complete MDL configuration can be found in Section 7.

We have implemented a parser for MDL, written in C, `lex` and `yacc`, which translates an MDL specification into an instance of the entity-relationship model. The model is expressed by a corresponding data-structure in C.

3.1 MDL for Simple Service Description

A `Service` is defined using the following syntax:

```
 <service-name> '=' protocol-base
  '[' <dest-port-range> {',' <src-port-range>}' ]'
 protocol-base = <protocol-base-name> || <protocol-base-no>
 <dest-port-range> = <port-range>
 <src-port-range> = <port-range>
 <port-range> = <port-number> ||
                <port-number> '-' <port-number> || '*'
```

The following code fragment defines the widely used services `smtp`, `ssh`, `https`, a service denoting all `tcp`-based packets, a service denoting all the well-known `tcp`-based services, and IPsec (ESP only):

```
SERVICES {
  smtp          = TCP [25]
  ssh           = TCP [22]
  https         = TCP [443]
  all_tcp       = TCP [*]
  well_known    = TCP [0-1023]
  ipsec50       =  50 [*]
}
```

Services can be grouped into a service-group by a statement of the following form:

```
 <s-grp-name> '='
   '{' <service-name1> { ',' <service-name2> } '}'
```

The following code fragment defines a service-group `gtwy_to_admin` by simple aggregation of services:

```
SERVICES {
  gtwy_to_admin = {ssh, https}
}
```

3.2 MDL for Advanced Service Description

Only the simple service definitions of Section 3.1 were available in *Firmato-v1* [Bartal et al. 1999]. Subsequently, however, it became clear that a richer set of definitions is necessary. Two extensions were added to the MDL syntax in *Firmato-v2*, dealing with multi-channel services and with connection-oriented `udp` and `icmp` services. These extensions are described in the next two sections.

3.2.1 *Multi-Channel Services.* One class of services we need to accommodate consists of multi-channel services, such as `ftp` and real media formats. The complexity of these services, from a filtering point of view, stems from the fact that they use dynamically allocated port numbers. A typical multi-channel service has a control channel that uses a fixed port number (e.g., `tcp` port 21 for `ftp`). Within the application-layer protocol running over the control channel, the communicating hosts dynamically agree on a port number for a data-channel, over which they transfer the bulk of their data.

This type of service creates difficulties, first and foremost, for firewall vendors. To support such services, vendors are forced to parse the application-layer protocols, keep additional states for such connections, and dynamically open the ports that were negotiated. Consequently, these services are pre-defined by the vendors as “atomic” or “built-in,” with dedicated syntactical constructs used to control them. For example, Cisco PIX Firewall can be configured to pass/block real media sessions by indicating the keyword `real-audio`.

In order to describe such services in *Firmato*, we extended the MDL for services to include a `predef-srv` attribute, as follows:

```
<s-grp-name> ‘=’
  ‘{’<service-name1> { ‘,’<service-name2> } ‘}’
  { ‘:’ <predef-serv> }
<predef-srv> = REAL-AUDIO || FTP || H323 || SQLNET || ...
```

The `predef-srv` attribute may be one of the specified keywords, identifying those services that have special handling by one or more firewall vendors, and which *Firmato*’s device-specific back-ends (see Section 5.3) know how to configure.

Note that this syntax allows a predefined service to have two alternative definitions: one definition using vendors’ special keywords (for those vendors that support the service), and one definition using the regular port numbers (for vendors who do not support the service).

The semantics are that *Firmato* will use the special vendor keywords if the device “understands” the service, and will fall back to the regular definition otherwise. Usually, the regular definition for such services is not accurate: either it does not describe the data channel at all, or it describes a large superset of ports that might be used by the data channel. This is the best one could hope for when configuring a device that does not have internal support for a multi-channel protocol.

3.2.2 *Connection-Oriented* `udp` *and* `icmp` *Services.* Most modern firewalls are stateful[4]: If the user writes a rule allowing a `tcp` connection from host a to host b, the firewall automatically allows the return packets, from b to a, that belong to the same `tcp` session, to get through the firewall. However, treatment of connectionless protocols (namely `udp` and `icmp`) is less uniform across different vendors.

Even though `udp` and `icmp` are defined as connectionless protocols, many applications use them with connection-like semantics. A typical example is the `ping` program. Issuing `ping` from host a to host b causes `icmp echo` packets to be sent from a to b, and `icmp echo-reply` packets to be sent back from b to a.

To simplify the configuration of such services in *Firmato-v2*, we added a REVERSE attribute to services within a service group. The semantics are that if the service group as a whole is used to configure traffic from a to b, the

[4]Cisco routers running IOS are an exception. Their basic and extended access lists are stateless.

REVERSE'd components will be configured with source and destination reversed: namely, from b to a. The following example shows the definition for the ping and real media service groups. Note that the definition of real_audio has both a predefined-service attribute (for those firewalls that recognize it), and a regular definition based on port numbers that utilizes the REVERSE attribute (for firewalls that do not support real_audio internally).

```
SERVICES {
  ping_req       = ICMP [8,0]
  ping_resp      = ICMP [0,0]
  ping           = {ping_req, ping_resp:{REVERSE}}
  r_tcp          = TCP [7070]
  r_udp          = UDP [6970-7170]
  real_audio     = {r_tcp, r_udp:{REVERSE}} : REAL-AUDIO
}
```

We note that *Firmato* comes bundled with an MDL file (def.srv) that contains most of the standard services defined in MDL.

3.3 MDL for Roles

The core of an organization's security policy is described by the roles it uses, and the relationships between role peers.

A peer relationshiop between two roles is defined by a statement of the following form, where the arrow defines the Direction attribute in an obvious way, the role-grp-name points to Peers, and the srv-grp-name points to a service-group:

```
 <role-(grp-)name> arrow <role-(grp-)name> ':' <srv-grp-name>
 arrow = '<-' || '->' || '<->'
```

Note that the three possible arrows are syntactical sugar. The left-arrow is equivalent to the right-arrow with the peer roles switched, and the double arrow is equivalent to two regular arrows.

The following code fragment defines the roles mail_server and internal_mail_server we discussed in Section 2.2. The roles gateway_in and gateway_out model the permissions of gateway interfaces in each direction.

```
ROLE_DEFINITIONS {
 mail_server          <-> *           : smtp
 internal_mail_server <-> mail_server : smtp
 gateway_in           <-  fw_admin    : admin_to_gtwy
 gateway_out           -> fw_admin    : gtwy_to_admin
 intranet_machine      -> all_tcp     : *
}
```

Roles are grouped into *OPEN* (default) role-groups by the following statement:

```
 <role-grp-name> '='
    '{' <role-name1> ',' <role-name2> ... '}'
```

Roles are grouped into *CLOSED* role-groups as follows:

```
 <role-grp-name> ‘=’
    ‘<<’ <role-name1> ‘,’ <role-name2> ... ‘>>’
```

The following code fragment defines the role-group gateway, bundling the unidirectional gateway roles into one role-group. Note that the gateway role-group is *CLOSED*, thus effectively "stealthing" hosts that assume this role-group.

```
ROLE_GROUPS {    # a closed group
  gateway   = <<gateway_in, gateway_out>>
}
```

3.4 MDL for Topology Description and Policy Mapping

The MDL syntax for topology description underwent a significant upgrade between *Firmato-v1* and *Firmato-v2*. Below we describe the details of the *Firmato-v2* MDL syntax.

3.4.1 *Gateways.* Gateways are defined by the following statement:

```
 <gateway-name> ‘=’ ‘{’
    <gw-interface-name1> ‘:’ ‘{’
        { ‘addr’ ‘=’ "<hwaddr>" } ,
        { ‘ip’ ‘=’ <IP-addr> } ,
        { ‘file’ ‘=’ "<filename>" } ,
        { ‘INVIS’ } , { ‘NO_GEN’ }
    ‘}’
    <gw-interface-name2>  ....
  ‘}’ ‘:’ <firewall-vendor> ‘{’
           ‘version’ ‘=’ <version-str>},
          ‘file’ ‘=’ "<filename>"
       ‘}’
 <hwaddr> = product-dependent interface address
 <firewall-vendor> = CKP || LMF || IOS || PIX
 <version-str> = product dependent version identifier
```

If the ip= keyword is used for an interface, a host-group with the name of the interface is created and instantiated with the provided IP address. Initially the interface's host-group has no roles associated with it. Note the ip= keyword is optional: firewalls that are layer-2 devices (such as the Lucent VPN Firewall Brick) do not have IP addresses associated with every interface. However, if no IP address is specified then the interface must have the INVIS attribute set, to signify that the interface is "invisible" (it's a layer-2 port).

The file= keyword tells *Firmato* where to output the rule-base. Note that it is possible to specify a file per interface or a file per gateway. The choice is vendor-dependent, some vendors require a single configuration file for the whole firewall while others accept separate files, one per interface.

A NO_GEN attribute tells *Firmato* not to generate any rules for the interface: The semantics are that all traffic will be allowed through his interface. This attribute is included to let the administrator avoid CPU bottlenecks. For instance, on a 2-interface firewall, filtering on both interfaces is almost entirely redundant. If the firewall is overloaded, the administrator can switch off the filtering on the inside interface, trading 50% of the firewall's CPU cycles against a somewhat elevated risk of insider attacks. In many cases the added risk is viewed as being much cheaper than buying a faster firewall. However, indiscriminate use of the `NO_GEN` attribute may completely invalidate the security policy: for example, if all the interfaces are marked `NO_GEN` then all traffic will be allowed.

The `firewall-vendor` keyword identifies the vendor and is used to select the appropriate vendor-specific back end (Section 5.3). "CKP" stands for Check Point FireWall-1, "LMF" for the Lucent VPN Firewall Brick (previously called Lucent Managed Firewall), PIX stands for the Cisco PIX Firewall, and IOS stands for a Cisco router running the IOS operating system.

The following code fragment defines the gateway `payroll_gw` as having `payroll_gw_e0/1` as its two interfaces. The gateway is modeled as a Cisco router running IOS version 12.0. We specify the hardware addresses of the two interfaces and their IP addresses; Cisco router interfaces cannot be invisible. A single configuration file controls all the router's interfaces, hence the `file=` keyword is associated with the gateway rather than with individual interfaces.

```
GATEWAYS {
 payroll_gw  = {
   payroll_gw_e0 : { addr="FastEthernet0/0",
                     ip=111.222.26.226}
   payroll_gw_e1 : { addr="FastEthernet0/1",
                     ip=111.222.24.210}
 } : IOS {version = 12.0,
          file="IOS_RULES_payroll_gw"}
}
```

3.4.2 *Zones.* Zones are defined via the following statement:

```
 <zone-name> '=' <host-grp-spec>
  ':' '{'<gtwy-interface-name1>
          {',' <gtwy-interface-name2>} '}'
  ':' (EXTERNAL || INTERNAL)

 <host-grp-spec> =
      '[' <ip-range> {',' <ip-range> } ']'           ||
      '{' <host-grp-name> { ',' <host-grp-name> } '}' ||
      <host-grp-name> '^' <host-grp-name>            ||
      '^' '{' <host-grp-name> { ',' <host-grp-name> } '}'

 <ip-range> = <IP-Addr> ||
              <IP-Addr> '-' <IP-Addr>  ||
              <IP-Addr> '/' <mask>
```

A zone definition implicitly defines a host-group, which has the same name as the zone, and contains the IP address ranges associated with the zone. Initially the zone's host-group has no roles associated with it.

IP addresses can be specified as individual IP addresses, as ranges, or as an address/mask pair. The address/mask syntax allows for shorter descriptions of subnets that are CIDR[5] blocks: for example, a class C subnet can be described as 192.168.1.0/24 rather than the equivalent but repetitive 192.168.1.0-192.168.1.255.

The difference operator "^" allows the user to specify "addresses that are in one host-group and not in another." The reason this operator was introduced was for the definition of the IP address ranges of the Internet: normally, the IP address space behind the firewall and in the DMZs[6] around it consists of multiple CIDR blocks that can be cleanly defined and are meaningful to the user. The rest of the Internet, however, is usually defined as "everything else." Using *Firmato-v1* syntax, users had to manually calculate the Internet zone's addresses. Furthermore, changing the subnets behind the firewall would have required changing the definitions of the appropriate subnet, *and* the definition of the Internet. In other words, the user would have had to change two or three places in the topology description files in a consistent manner, an error prone procedure.

The INTERNAL/EXTERNAL designation of a zone was introduced in *Firmato-v2* to provide a sense of direction to the network topology. Only one zone can be EXTERNAL, and it is the zone that is outside the organization's perimeter: the Internet zone. *Firmato* needs to know where the EXTERNAL zone is because some of *Firmato*'s vendor-specific back-ends require this information in order to produce correct configuration files (see Section 5.3).

The following code fragment first defines the zones `Z_payroll`, `Z_corp`, and `Internet`, each with associated IP addresses. This defines the three zones as host-groups. The code then defines parts of the network topology by specifying that the `Z_payroll` zone is connected to the `payroll_gw` gateway via the `payroll_gw_e1` interface. `Z_corp` is connected to `payroll_gw` via interface `payroll_gw_e1`, and also to the `ext_gw` gateway. The `Internet` zone consists of all other IP addresses, connects to interface `ext_gw_hme0`, and is designated EXTERNAL.

```
ZONES {
 Z_payroll = [ 111.222.26.0/24 ] : {payroll_gw_e0}
                : INTERNAL
 Z_corp  = [ 111.222.24.0/24 ] : {payroll_gw_e1, ext_gw_e1}
                : INTERNAL
 Internet  = ^ {Z_payroll, Z_corp} : {ext_gw_hme0}
                : EXTERNAL
}
```

[5]Classless Inter Domain Routing.
[6]DeMilitarized Zone.

3.4.3 *Host-Groups.* Host-groups are defined by the following statements. Note that a host is just a host-group containing a single IP address.

```
 <host-grp-name> { '=' <host-grp-spec> } { ':' <role-spec> }
 <role-spec> = <role-grp-name> ||
     '{' <role-grp-name> { ',' <role-grp-name> ... } '}'
```

The syntax for the `host-grp-spec` is the same as that used for zone definitions (Section 3.4.2). The `host-grp-spec` part of the definition is optional, since the host-group may have already been defined earlier, as a zone or interface. By omitting the `host-grp-spec` part, the user can assign a role to a zone or interface.

The `role-spec` part of the definition assigns one or more roles (or role groups) to the host-group. If multiple roles are specified, a *Firmato*-generated container role group is created and assigned to the host-group, saving the user from the need to define and name artificial role groups. The `role-spec` is optional too. If it is omitted then the host group does not have any roles associated with it. Note that omitting the `role-spec` does not mean that the host group cannot communicate at all: for example, the host group may be contained in a larger host group and inherit the latter's roles. A host group with no roles is useful, for instance, so the host-group can be used to define another host-group via the difference operator ^.

The following code fragment defines the hosts `dirty` (presumably outside the intranet) with the role of an external mail server. It assigns the role of a gateway to the payroll gateway's interfaces. Then it defines a host-group containing a subnet of all the hosts in research, and assigns an appropriate role.

```
HOST_GROUPS {
 dirty           = [ 111.222.100.6 ] : mail_server
 payroll_gw_e0                       : gateway
 payroll_gw_e1                       : gateway
 research        = [ 111.222.2.0/24 ]: corporate
}
```

3.4.4 *Changes from Firmato-v1.* The MDL syntax for topology description underwent a significant upgrade between *Firmato-v1* and *Firmato-v2*. The changes include: we removed the INTERFACES block and merged its content into the GATEWAYS block; we support hardware interface names; we allow "invisible" interfaces and interfaces for which rules will not be generated; the ZONES block now identifies a zone as either INTERNAL or EXTERNAL; a host-group is allowed to have multiple roles assigned to it, thereby creating an implicit (unnamed) role group; a host-group may have multiple IP address ranges, that can be described as unions and differences of either ranges or subnets of IP addresses; host-groups are no longer required to have a role: if no role is specified the host only assumes roles it inherits (if any).

4. TOPOLOGY-INDEPENDENT RULE COMPILATION

After the MDL parser converts the input files into an internal representation of the entity-relationship model, this representation needs to be translated into the appropriate firewall configuration files. The translation has to guarantee that the resulting files correctly implement the policy. This is done by the model compiler. The configuration files typically include services definitions, host-groups definitions, and rule-bases for each gateway interface. Obviously, the back-end of the compiler needs to be vendor-specific.

4.1 Rules for a Generic Firewall

We focus here on the intermediate rule-base generation. The compiler generates the rules in terms of an abstract, generic firewall, which uses an ordered rule list and disallows whatever is not explicitly allowed.

The generic rule format has the following fields: source host-group, destination host-group, service/service-group, action (pass/drop) and direction. The direction field is different from the role `Direction` attribute we had before. Most firewalls are also capable of filtering based on a *packet's* direction: which network interface card the packet is crossing, and whether the packet is crossing the interface from the network into the firewall or vice versa. We shall say more about the use of this field in Section 5.

When packets are filtered, the rules in the list are examined according to their order until a match occurs, and then the corresponding action is performed. The final rule in the list is always a default rule that drops every packet.

4.2 The Basic Model Compiler

The basic model compiler deals with translating an instance of the entity-relationship model into a firewall rule-base. The basic compiler ignores the network structure (i.e., gateway locations), and focuses on the definitions of roles, role-groups, and their assignments to host-groups. From these it deduces which pairs of host-groups should have a firewall rule that allows a certain service between them, ignoring the question of which gateway can actually enforce this rule. The output of the basic model compiler is therefore a single, centralized rule-base, which contains all the required rules to implement the policy. The centralized rule-base does not set the rule's direction field. This is achieved in the subsequent stage by the rule distributor (see Section 5).

The roles assigned to host-groups, and the relationships between peer roles, are a description of what operations are allowed between machines. It follows that the role-group assignment to a particular host-group H corresponds to a set of positive rules between H and other hosts assuming peer roles. We say that this set of rules is *associated* with H. If all the role-groups are *OPEN* then these positive rules are non-conflicting, hence form a correct rule-base.

The treatment of *CLOSED* role-groups is more involved. To illustrate this we will use a simple example. Consider a host h that assumes a *CLOSED* role-group C. Let H be a host-group that contains h and assumes a different role-group R. The fact that h is assigned a closed role-group implies that h should not inherit any roles from host-group H. However, if we apply our former strategy and generate the set of positive rules associated with H (as implied by R), these would *incorrectly* allow some services for h.

One way to get around this problem is for the compiler to split host-groups such that no resulting host-group includes hosts assuming *CLOSED* role-groups. In the example above, the compiler would replace H by $H' = H - \{h\}$, and H' would assume the role-group R. Then the compiler would generate the set of positive rules that is associated with H' (in place of H). This solution creates only positive rules. However, we view the creation of non user-defined host-groups as undesirable since they may make the resulting rule-base harder to debug.

Instead, our approach makes use of negative rules to avoid the need for new host-groups. Intuitively, we ensure that positive rules dealing with *CLOSED* role-groups appear *before* other rules in the rule-base, and these positive rules are followed by negative rules that capture the notion of "nothing else is allowed for the host-group." The rules that deal only with *OPEN* role-groups appear only after all the *CLOSED* role-groups have been dealt with. We call a host-group *CLOSED* if it is assigned with a *CLOSED* role-group and *OPEN* otherwise.

The rule generation algorithm is composed of the following 3 phases, at the end of which the default negative rule is added to the rule-base:

Phase 1:

```
foreach pair of CLOSED host-groups:
   generate all the positive rules between them;
   append the rules into the rule-base.
```

Phase 2:

```
foreach pair of a CLOSED host-group $H_1$ and an OPEN host-group $H_2$:
   foreach CLOSED host-group $G$ contained in $H_2$:
      generate negative rules between $H_1$ and $G$;
      append the negative rules into the rule-base.
   generate all the positive rules between $H_1$ and $H_2$;
   append the positive rules into the rule-base.
```

Phase 3:

```
         /* the ``nothing else'' rule */
foreach CLOSED host-group $H$:
   generate a negative rule between $H$ and ``*'';
   append it into the rule-base.
         /* the normal case */
foreach pair of OPEN host-groups:
   generate all the positive rules between them;
   append the rules into the rule-base.
```

As a basic component in the above description, the algorithm needs to generate, for a pair of host-groups H_1 and H_2, the list of positive rules that are associated with H_1 and apply to H_2. This is done as follows:

```
foreach role $r$ in the role-group assigned to $H_1$:
   foreach statement of the form {$r$ \$ $R:s$}:
         /* $R$=role-group; \$=direction; $s$=service; */
      if $H_2$ is CLOSED:   /* create a rule if $H_2$ has role $r$ */
         if the role-group assigned to $H_2$ contains a role in $R$
            create a positive rule between $H_1$ and $H_2$ with service=$s$
      otherwise for all host-groups $G$ that contain $H_2$:
         if the role-group assigned to $G$ contains a role in $R$
            create positive rule between $H_1$ and $H_2$ with service=$s$
```

Let us go back to the example above, where the *OPEN* host-group H contains the *CLOSED* host-group h. Assume that there is another open host-group H' also assuming role-group R and that R allows hosts to initiate a service to other hosts with the same role-group (i.e., there exists an MDL statement of the form "`R -> R : s1`"). Assume further that role-groups C and R do not have roles in common, and there are no definitions of the form "`C -> R : s2`"). We can see that only Phase 3 of the algorithm applies. It first generates a negative rule for h, followed by positive rules for the pair H and H', thus achieving the desired semantics.

A careful (informal) case analysis shows that our algorithm generates a rule-base that correctly implements the security policy defined by the model. Producing a formal, machine-checked, proof of the algorithm's correctness remains open for future work.

We make no claim regarding the length of the rule-base, and in particular the rule-base may contain redundant rules. We have made only straightforward optimizations, such as removing identical rules, to minimize the number of resulting rules. However, the rule distributor module (see Section 5), which was not part of *Firmato-v1*, offers significant improvements to the size of the rule-base distributed to each interface. The rule distributor ensures that a rule is assigned to a given interface only if it has a chance of being relevant to packets crossing that interface, based on the network's topology.

5. THE RULE DISTRIBUTOR

5.1 Overview

The basic model compiler generates a single list of logical rules. We then need to process this list in two ways.

First, we distribute the rules to each of the gateways in the network. To ensure that the security policy is observed, we place all the rules that concern a pair of host-groups on all the gateways along any possible routing path between them.[7] This is similar to Guttman's localization method [Guttman 1997]. We

[7]In principle, firewalls that are closer to the destination could have more open permissions, since "bad" packets are dropped by the first firewall on the path from source to destination. As more

also need to set the direction field on each rule. This field tells the gateway from which side of an interface it is supposed to expect a matching packet—entering the gateway from the adjacent zone or leaving the gateway onto the zone. These tasks are performed by the Rule Assignment and DIrection Setting (RADIS) algorithm (Section 5.2).

Second, we transform each assigned (logical) rule to a number of actual rules, conforming to the syntax and semantics of the actual filtering device (e.g., Cisco IOS, Lucent VPN Firewall Brick, etc). This is done by the vendor-specific back-ends (Section 5.3).

5.2 The Rule Assignment and DIrection Setting (RADIS) Algorithm

A naive way of distributing the rules is to replicate the central rule-base onto every interface, and set the direction of every rule to be "BOTH" (both incoming and outgoing directions are allowed). This was the approach we used in *Firmato-v1*. However, such a naive approach creates bloated rule-bases, which may lead to decreased performance of the filtering device; fewer rules usually imply faster processing.

Setting the direction to "BOTH" is also too open with respect to source address spoofing. Recall that the IP protocol has no way of verifying the source information in a packet. Suppose the central rule-base has a rule r allowing traffic from h_1 to anywhere. If genuine packets from h_1 are never routed through interface i in an "IN" direction, then setting r's direction to "BOTH" may allow an attacker to send spoofed packets (with a source field maliciously set to h_1) through i (cf. Wool [2004a]).

5.2.1 *Rule Distribution and the Routing Scheme.*

Clearly, an accurate assignment of rules to interfaces is related to the routing scheme used in the network. We are faced with two difficulties here. First, our basic model captures very little routing information. Second, the routing in the network may change dynamically, and as such is not easily described by a static model such as ours.

Instead of attempting to extend our model to capture the details of the routing, we chose to follow Guttman [1997] and use only a general and minimal assumption about the routing:

ASSUMPTION 5.1. *Packets are never routed in cycles.*

This assumption implies that a packet never passes through any interface or gateway more than once on its path from its source to its destination. For the case of a single gateway, Assumption 5.1 implies that bounce-routing[8] is forbidden. Note that the assumption does *not* imply that the network topology itself

open permissions can usually be written using fewer rules, this can be viewed as an optimization technique. We have not implemented such optimizations.

[8]Bounce routing is a routing anomaly in which packets from sources on subnet N go to a router R, which forwards them to another router R' that is *also* on subnet N. In such a case the packets would enter end exit R on the same interface. Bounce routing is usually indicative of a routing configuration error, since the hosts on subnet N can access the router R' directly without going through R.

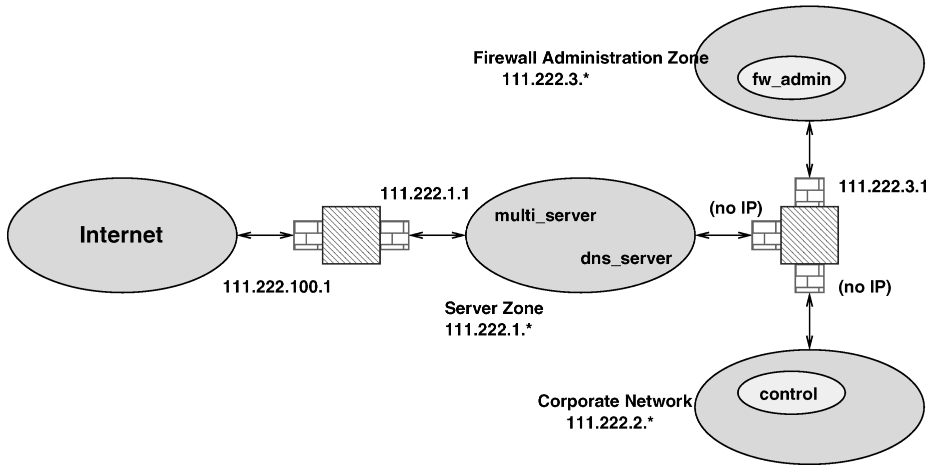

Fig. 3. An example network topology.

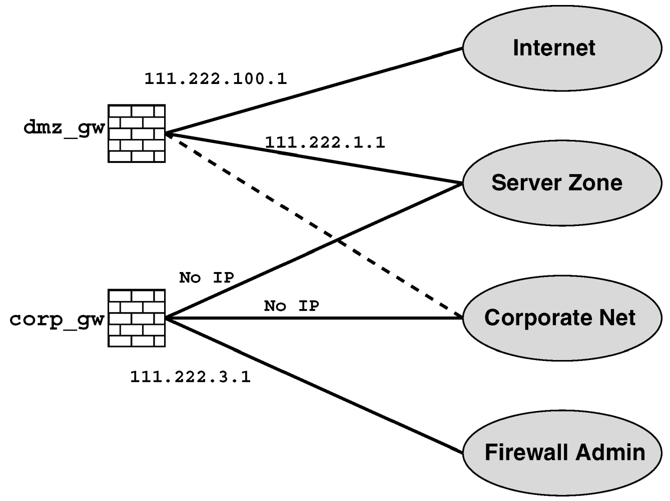

Fig. 4. A bipartite gateway-zone graph $H = ((G \cup Z), \mathcal{I})$ for the network topology of Figure 3, with interface edges shown as solid lines. G consists of two gateways, Z consists of four zones. The edges in $\mathcal{I}$ are labeled by the IP addresses of the interfaces they represent. Adding a third interface to dmz_gw, whose adjacent zone is the corporate net, would add the dashed line to the gateway-zone graph and create a cycle.

does not have cycles. This assumption, coupled with the topology knowledge in our model (limited as it is), allows us to compute fairly accurate rule-to-interface assignments.

5.2.2 *The Algorithm's Design.* We compute the rule assignment and direction setting by casting the question "does a rule r have a chance of ever being relevant to a non-spoofed packet attempting to pass through interface i in direction d?" in graph-theoretic terms. For this purpose we use the following auxiliary graph, which we call the gateway-zone graph (see Figures 3 and 4 for an example). This type of graph was also used by the localization method of Guttman [1997].

Definition 5.2. Let the *gateway-zone graph* be a bi-partite graph $H = ((G \cup Z), \mathcal{I})$ whose vertices consist of the set of gateways G and the set of zones Z. The set of interfaces $\mathcal{I}$ form the edges: H contains an edge $i = (g, z)$ connecting a gateway $g \in G$ to a zone $z \in Z$ iff g has an interface i whose adjacent-zone is z.

Consider a rule r with a source host-group s and a destination host-group d. Assume for simplicity that both s and d do not span more than one zone each. Let z_s and z_d be the zones in which s and d reside, respectively. Since packets are not routed in cycles (Assumption 5.1), it suffices to place r on interface $i = (g_i, z_i)$ if and only if a simple path[9] from z_s to z_d via i exists in the gateway-zone graph H. To force a simple path to go through the edge i, we need to delete the edge $i = (g_i, z_i)$ from H, and look for *two* edge disjoint paths connecting the path endpoints to g_i and z_i; the full path would consist of the path from one endpoint to g_i, the edge i, and the path from z_i to the other endpoint.

More precisely, if there exist two edge-disjoint simple paths in H: one from z_s to g_i, and the other from z_i to z_d, then we need to place r on interface i with direction "OUT", since the traffic will flow from the gateway outward to the adjacent zone. Similarly, if there exist two disjoint paths, one from z_s to z_i and the other from g_i to z_d, we need to place r on i with direction "IN".

Finding two edge-disjoint paths in a graph is a special case of the multi-commodity flow problem: Each source-destination pair corresponds to one commodity, and each edge has capacity 1. Unfortunately, the multi-commodity flow problem is NP-complete, and remains NP-complete even if there are only two commodities, on an undirected graph, with edge capacities which are all 1 (see Problem ND39 in Garey and Johnson [1979]). So the approach seems difficult to implement.

Luckily, we can relax the requirements slightly and obtain an efficient algorithm. Our strict requirements are to test the existence of two disjoint paths, one between z_s and g_i, and the other between z_i and z_d. Instead, we test whether two edge-disjoint paths exist, between the *pair* of sources $\{z_s, z_i\}$ and the *pair* of destinations $\{g_i, z_d\}$. This relaxed test is a special case of a maximal flow problem in a unicost network, and can be solved in polynomial time using the augmenting paths algorithm (cf. Ahuja et al. [1993]).

If the flow algorithm returns that the maximal flow is less than 2, then we conclude that the two edge-disjoint paths we care about do not exist. So the relaxed flow algorithm does not have "false negatives." However, if the maximal flow is equal to 2, then there could be two cases:

(1) Disjoint paths exist that connect $z_s \leftrightarrow g_i$ and $z_i \leftrightarrow z_d$.
(2) Disjoint paths exist that connect $z_s \leftrightarrow z_d$ and $z_i \leftrightarrow g_i$.

[9]A path in a graph is called *simple* if it has no loops: It does not go through any vertex more than once.

Case 2 is a "false positive," since the paths that were found do not have the desired structure, and cannot be spliced with the edge $i = (g_i, z_i)$ to form a simple path from z_s to z_d through i.[10] Thus, using the relaxed flow algorithm, we may place a rule on an interface when it is redundant there.

It is easy to see that if the gateway-zone graph contains no cycles, then the relaxed flow algorithm never makes mistakes. If there are no cycles in H, then only one path exists between g_i and z_i, namely, the edge $i = (g_i, z_i)$ itself. Removing the edge i ensures that case 2 can not occur. More generally, a moment's reflection shows that if the edge $i = (g_i, z_i)$ is "critical" (removing it from H disconnects the gateway-zone graph into two connected components), then the relaxed flow algorithm will not make a mistake with respect to the edge i. These are important special cases because actual network topologies tend to be tree-like, with very few cycles (if any). In particular, the gateway-zone graph for a single firewall and its adjacent zones is always a tree.

5.2.3 *Efficient Implementation of the RADIS Algorithm.* As we have discussed, we need to solve a maximal flow problem on a unicost network, using an augmenting-paths algorithm. Furthermore, we only need to find two disjoint paths. Therefore, the augmenting paths algorithm has a very simple, 2 step, form. To test whether a rule should be placed on interface $i = (g_i, z_i)$ with direction "OUT", we need to (a) find a simple path from z_s to g_i, then (b) find an augmenting path from z_i to z_d. The inbound case is similar. Finding a single path is linear in the size of the graph, so the RADIS algorithm runs in $O(|\mathcal{I}|)$ time per rule per interface.

We can simplify the algorithm further, since we only have a single step in which an augmenting path is computed. We achieve this by treating H as a *directed* graph H' which has 2 parallel edges for every interface $i = (g, z)$: the edges (g, z) and (z, g). Step (a) is implemented as a breadth-first search (BFS) on H', and the augmenting step (b) becomes another BFS on H', that is restricted not to use any directed edge that step (a) already used. If step (a) uses a directed edge (g, z), then step (b) is allowed to use the reverse-direction edge (z, g)—this would correspond to an augmenting path being found. Finally, here is the pseudo code for the RADIS algorithm:

```
(1) Input: a rule $r$ with source in zone $z_s$ and destination in zone $z_d$, and an
    interface $i$.
(2) Remove edge $i = (g_i, z_i)$ from $H$.
(3) Create the graph $H'$: $H'$ has two parallel directed edges for each undirected
    edge in $H \setminus \{i\}$.
(4) Outbound: Look for a path $z_s \leftrightarrow g_i \rightarrow z_i \leftrightarrow z_d$, traffic flows from the
    gateway to the zone.
    (a) Do a BFS on $H'$ from node $z_s$ until node $g_i$ is found; return "false" if no
        such path exists.
```

[10]Note that the 2nd case is not trivially true because we removed the edge $i = (g_i, z_i)$ from the gateway-zone graph H.

Table I. Firewall Features that Differ Between Vendors

	Lucent VPN Firewall Brick	Check Point FireWall-1	Cisco PIX Firewall	Cisco IOS Router
Names	Yes	Yes	Some	No
Groups	Yes	Yes	Some	No
IP Ranges	Yes	Yes	No	No
Stateful	Yes	Yes	Yes	No
Trust Levels	No	No	Yes	No
Directional	Yes zone-centric	GUI: No INSPECT: Yes	Yes via commands	Yes device-centric
Default Stance	Drop	Drop	Inbound: Drop Outbound: Pass	Drop
Predefined Services	No	Yes	Yes	No
Layer	layer 2	layer 3	layer 3	layer 3

(b) Do a BFS on H' from node z_i until node z_d is found, but avoid the edges already used in (a). Return "false" if no such path exists.
(c) If (a) and (b) return "true" then the flow = 2: Place rule r on interface i with direction "OUT".

(5) Inbound: Look for a path $z_s \leftrightarrow z_i \rightarrow g_i \leftrightarrow z_d$; traffic flows from the zone to the gateway.
(a) Do a BFS on H' from z_s until node z_i is found; return "false" if no such path exists.
(b) Do a BFS on H' from g_i until node z_d is found, but avoid the edges already used in (a). Return "false" if no such path exists.
(c) If (a) and (b) return "true" then the flow = 2: Place rule r on interface i with direction "IN".

5.3 The Back-Ends: Generating Device Dependent Rules

The last step of the rule distributor is to generate a rule-base for each interface i, in the vendor's configuration language. The rule-bases are written into one or more files, either one file per gateway or a file per interface (the choice is vendor-specific).

Firmato-v1 only had a single back-end, which supported the Lucent VPN Firewall Brick. In *Firmato-v2* additional back-ends were added, to support Check Point FireWall-1, Cisco PIX Firewall, and Cisco IOS (router access lists). These back-end modules encapsulate the knowledge of the vendor's syntax and semantics, and translate the generic rules produced by the *Firmato* compiler into the vendor's language. Obviously, this type of conversion requires in-depth device-specific knowledge, so we omit many of the details. We only touch upon several issues that had to be addressed in the back-ends and highlight some of the semantic differences between *Firmato*'s model and the various products we support. See Table I for a comparison table of the features that differ among vendors.

5.3.1 *Names and Groups.* Both the Lucent VPN Firewall Brick and FireWall-1 accept named hosts and services, and allow both types of entities to be grouped, so this aspect of *Firmato*'s model could be translated relatively easily. Cisco's products (both Cisco PIX Firewall and Cisco IOS) have very minimal support for names: only an individual IP address may have a name and there is no concept of a named group.[11] Thus *Firmato*'s back-ends expand the source, destination, and service components of each rule into their basic IP addresses, protocols, and port numbers. In case of grouped definitions, *Firmato* produces one Cisco rule for each combination of basic source, destination, and service values, from the cross-product of the source, destination, and service groups appearing in a generic rule.

5.3.2 *IP Address Ranges.* MDL allows the definition of host-groups consisting of multiple arbitrary ranges. Both FireWall-1 and the Lucent VPN Firewall Brick support arbitrary IP ranges. Cisco IOS and PIX only accept host-groups that are CIDR blocks, described by an IP address and subnet mask (e.g, 111.222.1.0/24). Thus *Firmato*'s back-ends convert each MDL host-group into the smallest number of CIDR blocks that exactly cover it. In the worst case, a single range (0.0.0.1−255.255.255.254) can only be covered by 62 separate CIDR blocks. Therefore, a single *Firmato*-generated rule can produce many IOS ACL (access control list) or PIX rules—the number would be the product of the number of CIDR blocks in the rule's source and the destination.

5.3.3 *Statefulness.* Check Point FireWall-1, Cisco PIX Firewall, and the Lucent VPN Firewall Brick are all stateful: A rule allowing a `tcp` connection from host a to host b, automatically allows the return packets, from b to a, that belong to the same `tcp` session, to get through the firewall. All three products are stateful for `udp` as well (despite the fact that `udp` does not officially have session semantics). Cisco IOS is stateless, thus the *Firmato* back-end has to produce two IOS access list statements for each *Firmato* rule, one for the initiating packets and one for the returning packets.

Note that the IOS statement allowing the return traffic is very open: `tcp` source ports are dynamically chosen, so as to allow return traffic (in which source and destination port numbers are swapped), the generated rule allows *any* destination port, which is clearly more than what should be allowed. We mitigate some of this exposure by using Cisco's `established` keyword: this keyword requires returning packets to belong to a pre-existing `tcp` session. However, without keeping state, the router can only enforce the `established` keyword by checking that the `syn` and `ack` header flags are set—which can be circumvented by a malicious adversary. Fundamentally, a stateless packet filter is simply too weak to correctly enforce most reasonable security policies.

5.3.4 *Trust Levels.* Cisco PIX Firewall assigns a trust-level to each firewall interface, and the syntax for controlling outgoing traffic (going from a high-trust interface to a low-trust interface) is radically different from that to control inbound (low-to-high) traffic. To support this, the *Firmato-v2* model was extended so a zone could be tagged as EXTERNAL or INTERNAL (recall Section 3.4.2). *Firmato*'s PIX back-end computes the interface trust-levels as

[11]Cisco PIX Firewall v6.2, introduced in 2002, added support for named groups. The new syntax is not generated by *Firmato*.

follows. It performs a breadth-first search (BFS) on the gateway-zone graph, starting with the (unique) zone that is marked EXTERNAL, and calculates the BFS depth of each zone. The interfaces receive trust-levels scaled such that the interface connecting the zone at depth 0 (the EXTERNAL zone) to its adjacent gateway has trust-level 0, and (one of) the interface(s) connected to the maximal depth zone receives trust-level 100. Each interface receives a distinct trust-level.

5.3.5 *Directionality.* Firewall vendors deal with rule directions (into or out of an interface) in different ways [Wool 2004a].

The easiest back-end in this respect was Cisco IOS, which is completely straightforward. Almost as easy was the Lucent VPN Firewall Brick; the only twist is that the Lucent VPN Firewall Brick uses zone-centric direction-names (i.e., traffic entering an interface is going "OUT" in Lucent VPN Firewall Brick parlance).

Cisco PIX Firewall uses different commands for each direction, based on the interface trust-levels. Thus the PIX back-end first computes whether a particular rule is outgoing or incoming (according to the trust-levels) and outputs the appropriate PIX commands.

Check Point FireWall-1 does not support rule directions at all through its GUI (whose file format is undocumented) but *does* support rule directions in the documented (but low level) INSPECT language. The *Firmato-v2* FireWall-1 back-end outputs the INSPECT language, thus it is able to generate rule directions—but as a result, the generated FireWall-1 configuration is incompatible with parts of the GUI.

5.3.6 *Default Stance.* Most firewall products have a default stance of "drop any traffic that is not explicitly permitted." A surprising exception is Cisco PIX Firewall. For outgoing connections, PIX's default is "pass everything that is not explicitly denied." This stance conflicts with the *Firmato* model, thus we needed to reverse the PIX default for outgoing traffic. Up to PIX version 4.4, changing this default behavior was possible only in very limited ways: apparently it is not possible to change the default to "drop" and then to produce PIX commands that allow traffic with all three fields (source, destination, and service) specified to be different from "*". Cisco added significant functionality to PIX version 5.0, including changing the default stance to be "drop" on every interface that is configured with the new syntax. We have not upgraded the *Firmato* PIX back-end to support v5.0 (and above) PIX commands yet.

5.3.7 *Predefined Services.* There is a large degree of variability in services that are predefined by different vendors. Therefore, in *Firmato*'s default services MDL file (recall Section 3.2.2) we set the predefined attribute for 40 services, that were predefined by at least one vendor.

Check Point FireWall-1 ships with a large number of predefined services (over 100). However, many of these services are regular `tcp` or `udp` services, whose definition only describes the port numbers the service uses. *Firmato* declares as predefined only those services whose FireWall-1 definition is more sophisticated: multi-channel protocols. For these services, the *Firmato* FireWall-1 back-end generates references to the Check Point definitions.

The Lucent VPN Firewall Brick does not have any special syntax for predefined services. Instead, its kernel traps those multi-channel services it supports (based on their control-channel port number), and automatically activates the service-specific filtering code. Cisco IOS has minimal support for complex services. Therefore, *Firmato*'s Lucent VPN Firewall Brick and Cisco IOS back-ends ignore all the predefined service attributes an always generate regular (port-number-based) service definitions.

Cisco PIX Firewall recognizes the names of predefined services within its filtering rules, but only as mnemonics for port numbers (e.g., the keyword `ftp` is equivalent to the number 21). Therefore *Firmato* does not output predefined names in the filtering rules. However, Cisco PIX Firewall does support multi-channel services via a separate `fixup` command, which the *Firmato* PIX back-end produces.

6. THE RULE ILLUSTRATOR

The rule illustrator translates the firewall rule-base (i.e., the outcome of the model compiler) into a graph-based representation that visualizes both the host-group's structure and the services (packets) that the firewall passes. The illustrator creates a visualization of the policy as it is seen from the point of view of a single gateway interface: it displays which host-groups are on which side of the interface, and the firewall rules enforced by this interface.

The rule illustrator makes the task of the debugging of the *Firmato* compiler easier: it is clearer to look at a colorful graph than to sift through long, automatically-generated rule-bases in arcane firewall format. However, the illustrator is also useful in its own right. For instance, since it reads the the firewall's rule-base, it can be used to reverse engineer existing rule-bases in order to extract the policy from them.

The rule illustrator was implemented in C, and uses a graph layout tool. In *Firmato-v2* the graph layouts are produced by `dot` [Gansner et al. 1993; Dot 2001]. In parallel with the development of *Firmato-v2*, the illustrator was integrated into the Fang firewall analysis prototype [Mayer et al. 2000], which evolved into the Lumeta Firewall Analyzer [Wool 2001]. An example of the illustrator's `dot`-generated output, visualizing the rules of an example configuration, appears in Figure 5.

6.1 The Host-Group Structure

The first task of the illustrator is to visualize the structure of the host-groups with respect to containment and intersection. This is important since a rule that applies to some host-group A is inherited by any host whose IP address falls within A.

We display the host-group structure as a graph whose nodes are labeled by the host-group name. A solid black edge between two nodes A and B indicates that one contains the other. The direction of containment, whether $A \subset B$ or $B \subset A$, is indicated by which node is above the other (see below). A dashed

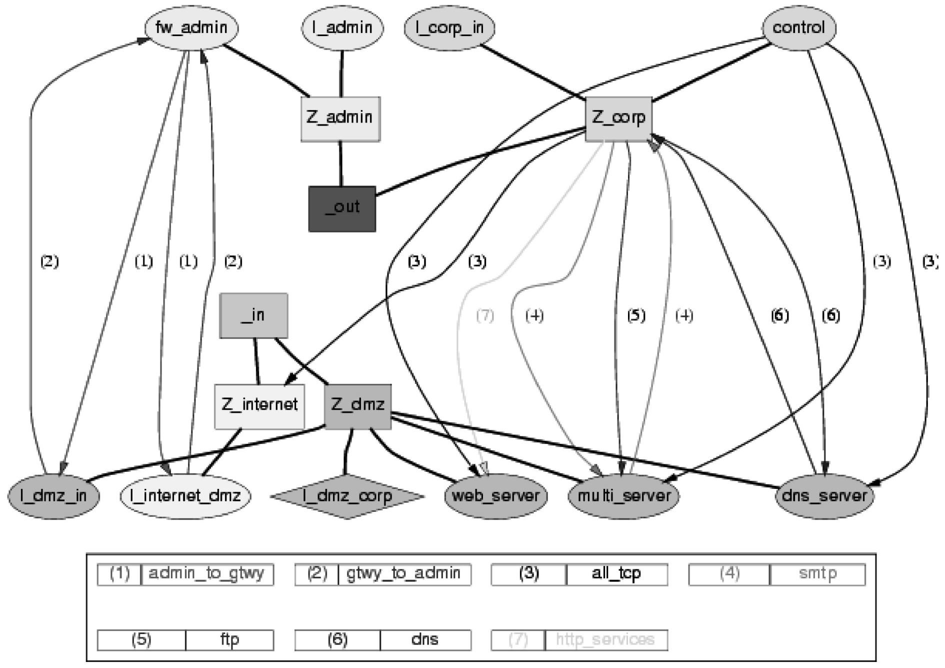

Fig. 5. The output of the Rule Illustrator, from the point of view of the `I_dmz_corp` interface.

black edge indicates host-groups that intersect: they overlap but one does not completely contain the other.

We first partition the host-groups into two categories, depending on the side of the interface in which they reside.[12] One category is called the "outside": this is the category that contains the zone directly adjacent to the interface we are illustrating. The other category is called the "inside."

We visualize this partition by introducing two artificial host-groups, called `_in` and `_out`, and displaying them as two rectangle-shaped nodes in the middle of the graph (host-groups of zones are also shown as rectangles, the interface for which the illustration is being created is shown as a diamond; other host-groups are shown as ovals). Then we display the "inside" host-groups as a tree that grows downward from the `_in` node, and the "outside" ones as a tree growing upward from the `_out` node. Thus for "inside" host-groups, if A has an inclusion edge to B and A is above B (A is closer to `_in`) then $A \supset B$. For "outside" host-groups the group closer to `_out` includes the other. The trees represent the minimum inclusion relation whose transitive closure equals the host-groups inclusion relation. The layout of the trees is determined by the inclusion relation; intersection edges, which do not obey the tree layering, are added later.

Finally, we assign colors to the nodes to represent the zones: all the host-groups belonging to the same zone get the same color.

[12]We are implicitly assuming that the zone-gateway graph is acyclic otherwise a host-group can simultaneously be on both "sides" of an interface.

Remark. In its current state the illustrator is unable to visualize host-groups that are both "inside" and "outside" at the same time. We did not find a visually-pleasing way to do this, although artificially breaking the group into to in- and out-subgroups and displaying each separately is a partial solution. Such a zone-crossing host-group may be useful, say, to describe all the corporate hosts, which reside in two subnets that are separated by a firewall gateway.

6.2 Adding the Rules

The illustrator displays the rules only for services that cross the interface—other rules (dealing with services where both end-points are on one side of the interface) are ignored.

The rules are represented by directed edges (arrows) from source to destination. An edge from A to B represents a service that the firewall allows to pass from host-group A (and its sub-groups) to host-group B (and its sub-groups). Different services are shown by color coding the edges, for example, `all_tcp` is a red arrow, `telnet` is a blue arrow, and so on.

Remark. In its current state the illustrator is unable to visualize negative ("drop" action) rules. This has not been a major problem so far since the rule-bases we have seen tend to have a single "drop everything" rule as a default, and multiple "pass" rules for allowed services. Moreover, the *Firmato* compiler generates negative rules only when *CLOSED* role-groups are defined, and then these "drop" rules are there to ensure that a host-group does not inherit permissions of other host-groups that contain it. Thus ignoring these negative rules has minimal effect on the meaning of the graph.

7. A COMPLETE EXAMPLE

In this section we show a complete annotated example, which illustrates our methodology and tools. For this purpose we consider an imaginary (yet realistic) corporation with a two-firewall network configuration, as shown in Figure 3. For concreteness we list the corporation's various IP addresses; however, they bear no relationship to the real `111.222.*.*` subnet (if one exists at all).

7.1 The Environment

The external firewall, which guards the corporation's Internet connection, is a Cisco PIX Firewall. Behind it is the DMZ, which contains the corporation's externally visible servers. In our case these servers provide `http`/`https` (web), `ftp`, `smtp` (e-mail), and `dns` services. The corporation actually only uses three hosts to provide these services, one for `http`/`https`, one for `dns`, and the last (called `multi_server`) for all the other services.

Behind the DMZ is the internal firewall, a Lucent VPN Firewall Brick, which guards the corporation's intranet. This firewall actually has three interfaces: one for the DMZ, one for the corporate network zone, and a separate interface connecting to the firewall administration host. Within the corporate network zone, there is one distinguished host, `control`, which provides the administration for the servers in the DMZ.

7.2 Policy Goals

The policy we consider is a rather simple one, which nonetheless covers many of the aspects that occur in more complex, real-life policies. Its premise is that internal corporate users are basically trusted and thus are relatively unrestricted, whereas external users are allowed access only to content that is explicitly made public. In more detail, the policy has the following goals:

(1) Internal corporate hosts can access all the resources on the Internet.
(2) External hosts can only access the servers in the DMZ. In particular, `smtp` sent to corporate users is only allowed via the `mail_server`, and `dns` services are provided to the Internet only by the `dns_server`.
(3) The DMZ servers can be updated only by the web administrator host `control`. Other corporate hosts have the same privileges as Internet hosts with respect to the DMZ servers.
(4) The firewall gateway interfaces are only accessible from the `fw_admin` host and are otherwise inaccessible to any host (this practice is usually called "stealthing" the gateways).

7.3 Role Declarations and Definitions

We now need to write the MDL to implement our policy. The first step is to define the various services that we deal with such as `smtp`, `http`, and so on. In the interests of brevity, we omit the straightforward details of the service definitions. The next step is to define the various roles and the relationships between them. We use the following as our basic roles.[13]

```
ROLES {
 R_admin   # firewall administration
 R_interface  # gateway interfaces
 R_internet R_corporate R_mail_server R_web_server
 R_dns_server R_ftp_server R_web_admin
}
```

Next we define role-groups. The only role group we need to define is `R_gw`: it is almost identical to the `R_interface` role, except that it is a *CLOSED* role-group. Goal (4) requires us to "stealth" the gateways, so we use a *CLOSED* role-group to ensure that gateway interfaces do not inherit any roles associated with, say, the corporate hosts.

```
ROLE_GROUPS {
 R_gw   = <<R_interface>>
}
```

Once the roles are declared, we need to define the relationships between them, as follows. Firewall administration requires two definitions that relate the administrative role `R_admin` to the gateway interfaces. We need two definitions since the services are asymmetric.

[13]As a convention we attach an "R_" prefix to all the role names, but this has no syntactic meaning.

```
ROLE_DEFINITIONS {
 R_admin -> R_gw : admin_to_gtwy
 R_admin <- R_gw : gtwy_to_admin
```

Corporate hosts are allowed access to the Internet. However, they do not gain access to the gateway interfaces since the `R_gw` group is *CLOSED*.

```
 R_corporate -> R_internet : all_tcp
```

The `R_web_admin` role is the only one that can update the servers in the DMZ.

```
 R_web_admin ->  R_mail_server : all_tcp
 R_web_admin ->  R_web_server  : all_tcp
 R_web_admin ->  R_dns_server  : all_tcp
 R_web_admin ->  R_ftp_server  : all_tcp
 R_web_admin ->  R_dns_server  : all_tcp
```

Finally, we complete the role definitions by defining the DMZ roles. E-mail in both directions must go through the `mail_server`, and `dns` does not go into the corporate net directly, but only through the `dns_server`. `ftp` and `http` services from the Internet and from the corporate net are allowed to the appropriate servers.

```
 R_mail_server <-> R_internet  : smtp
 R_mail_server <-> R_corporate : smtp

 R_dns_server  <-> R_internet  : dns
 R_dns_server  <-> R_corporate : dns

 R_ftp_server  <-  R_internet  : ftp
 R_ftp_server  <-  R_corporate : ftp
 R_web_server  <-  R_internet  : http_services
 R_web_server  <-  R_corporate : http_services
}
```

Remarks.

—So far we have not used the network topology at all. All we needed to consider up to this point were the applications that the organization uses. Thus the same definitions can be used at other corporate sites that wish to implement the same policy; such sites will only need to write the MDL sections that deal with topology.

—There is a special system-defined role-group with the name "*", which is shorthand for "all roles." However, we chose not to use it, and instead opted for defining a special `R_internet` role. Had we used "*" instead of `R_internet` (i.e., `R_corporate -> * : all_tcp`), as a side effect we would have given hosts with the role `R_corporate` the ability to update the DMZ servers, since the "*" role contains all the DMZ server roles.

—Using the "*" role has another, more subtle problem. The "*" role eventually translates to "all IP addresses," and this is a host-group that *spans all the zones*, for example, hosts in this group reside on both sides of the external gateway. Zone-spanning is a very common mis-configuration in firewalls [Wool 2004b], which makes the firewall susceptible to spoofing attacks: If a host-group resides on multiple "sides" of the firewall, including on the Internet side, the firewall cannot distinguish between legitimate traffic with internal-looking addresses arriving on the external interface and spoofed traffic arriving on the external interfaces. Therefore as a precaution we advocate being as specific as possible. Following this guideline, in the sequel we assign the role `R_internet` precisely to the external IP addresses, and to nothing else.

7.4 Network Topology and Assignment of Roles

The last part of the MDL is topology-specific. It involves defining the various host-groups, gateways, interfaces, and zones of the corporate network, and assigning roles to the host-groups.

We first define the two gateways: the external gateway as a Cisco PIX Firewall, and the internal gateway as a Lucent VPN Firewall Brick. Then we define the zones between and around the gateways. Note the compact definition of the Internet zone `Z_internet` using the difference operator, and its designation as EXTERNAL.

```
GATEWAYS {
dmz_gw  =
  {
    I_internet_dmz : {addr="ether0", ip=111.222.100.1},
    I_dmz_in       : {addr="ether1", ip=111.222.1.1}
  } : PIX {version = 5.2, file="PIX_RULES_dmz_gw"}

corp_gw =
  {
    I_dmz_corp : { addr=ether0, INVIS,
                   file="RULES_I_dmz_corp" }
    I_corp_in  : { addr=ether1, INVIS,
                   file="RULES_I_corp_in" }
    I_admin    : { addr=ether2,
                   ip = 111.222.3.1, file="RULES_I_admin"}
  } : LMF
}

ZONES {
Z_dmz      = [111.222.1.0/24] : { I_dmz_in, I_dmz_corp }
       : INTERNAL
Z_corp     = [111.222.2.0/24] : { I_corp_in }
       : INTERNAL
Z_admin    = [111.222.3.0/24] : { I_admin }
       : INTERNAL
Z_internet = ^{Z_dmz,Z_corp,Z_admin} : { I_internet_dmz }
       : EXTERNAL
}
```

Next we assign roles to two of the zones, and to the gateway interfaces. We choose not to give the server zone or the firewall administration zone any roles since the machines inside each of these zones play different roles. We assign the role R_gw to the three gateway interfaces that have IP addresses. Recall that this is the *CLOSED* role-group; if we had used the similar *OPEN* role R_interface, we would have compromised the stealthing of the gateways. We do not assign roles to interfaces I_dmz_corp and I_corp_in: They do not have IP addresses (being invisible) and thereby cannot be referred to by IP packets.

```
HOST_GROUPS {
Z_corp     : R_corporate
Z_internet : R_internet

# The gateway interfaces.
I_internet_dmz : R_gw
I_dmz_in       : R_gw
I_admin        : R_gw
```

Finally we define the special machines in the network: the DMZ servers web_server, multi_server and dns_server, the firewall administration machine fw_admin, and the web administration machine control. Note that the multi_server has several roles: this fact is topology-specific and is not visible in the role definitions we showed in Section 7.3. The same role definitions could have been used without change if the various functionalities were split over several machines.

```
multi_server   = [111.222.1.17]  : {R_mail_server, R_ftp_server}
web_server     = [111.222.1.17]  : R_web_server
dns_server     = [111.222.1.10]  : R_dns_server
fw_admin       = [111.222.3.7]   : R_admin
control        = [111.222.2.54]  : R_web_admin
}
```

7.5 Compiling and Visualizing the Rules

We ran the *Firmato* compiler on the example files we discussed in Sections 7.3–7.4. The compiler generated 42 rules, 10 of which were negative due to the *CLOSED* R_gw role group. Here is a sample of the device-specific generated rules. These are in the Lucent VPN Firewall Brick configuration language, intended for the I_dmz_corp interface, connecting the internal firewall to the DMZ zone in front of it, and ultimately to the Internet (recall Figure 3).

```
 (rule_number=1,
    src_host=fw_admin, dst_host=I_dmz_in,
    service=admin_to_gtwy,
    action=pass, direction=IN)
 (rule_number=12,
    src_host=*, dst_host=I_dmz_in, service=*,
    action=drop, direction=BOTH)
```

```
(rule_number=21,
   src_host=Z_corp, dst_host=Z_internet, service=all_tcp,
   action=pass, direction=IN)
```

Note that the direction field in the rules has been set by the Rule Distributor (Section 5). The traffic controlled by rules 1 and 21 is leaving the internal firewall into the DMZ zone via the `I_dmz_corp` interface, so in *Firmato*'s model the traffic has a direction of "OUT". However, in the Lucent VPN Firewall Brick language this traffic's direction is called "IN" (into the adjacent zone), so the vendor-specific back-end reversed the direction names.

We already saw the output of the illustrator in this configuration: Figure 5 shows a graphic representation of the generated rules from the point of view of the `I_dmz_corp` interface. As we discussed in Section 6, the figure separates the host-groups into two classes depending on their location with respect to the interface, and shows the inclusion relations between host-groups. Zones are shown as rectangles, the interface for which the illustration is performed is shown as a diamond, and all other host-groups are shown as ovals. For example, the class `_out` includes the zones `Z_admin` and `Z_corp`, zone `Z_admin` includes the host `fw_admin` and the interface `I_admin`, and so forth. The services are coded by color and have a number that corresponds to the legend. For example, the black edge going down from `Z_corp` to `Z_Internet`, with number (3), corresponds to the `all_tcp` service.

8. FIELD DEPLOYMENT

The *Firmato* system was used to configure the operational firewall protecting the Bell Labs Research network for a period of several months starting in November 1999. The main firewall was a Lucent VPN Firewall Brick with four interfaces. A Cisco router was located in front of the firewall, which also acted as a filtering device using access control lists. The field deployment was planned jointly with the Bell Labs' firewall administration team.

Our first goal was to configure the Lucent VPN Firewall Brick using *Firmato*. In preparation for the transition to *Firmato*, we needed to convert the existing firewall policy to an equivalent MDL representation. This was done manually, by inspecting the firewall rules and distilling the appropriate roles and host-groups from them. The firewall had under 50 rules so a manual conversion process was reasonable. A firewall with thousands of rules, such as those described in Wool [2004b], would have probably required the development of automated conversion tools, followed by manual correction.[14]

In addition, we needed to write the MDL describing the network topology. This was also done by a manual process, using the existing routing tables as a starting point. Feedback from the firewall administration team during this planning phase was the main driving force behind the changes in the network topology MDL between *Firmato-v1* and *Firmato-v2*.

[14]Converting legacy firewall policies from one language to another (MDL or another vendor's language) is an interesting area for future research.

In order to protect the network from firewall configuration mistakes that might occur during the transition, the firewall administration team placed a new firewall *in front* of the existing firewall. The new firewall was configured by *Firmato*. Since it was located between the legacy firewall and the Internet, the new firewall was supposed to block all unauthorized inbound traffic, and so the legacy firewall would only see authorized inbound traffic. However, in case of a configuration problem in the new firewall, the network was still protected by the legacy firewall. Therefore the administrators instrumented the legacy firewall to log any inbound traffic that it dropped, and used this log to verify that the *Firmato*-generated configuration was working as designed. Once the firewall administration team gained confidence in *Firmato*, they planned to decommission the legacy firewall and redeploy it elsewhere. Note that the Lucent VPN Firewall Brick is a layer-2 device, so inserting an extra Lucent VPN Firewall Brick in front of the legacy firewall did not require re-assigning the IP addresses on the adjacent subnet (no artificial transit networks had to be created), nor were any routing changes needed [Limoncelli 1999].

We discovered that loading *Firmato*-generated rules onto the Lucent VPN Firewall Brick was not entirely trivial. Lucent VPN Firewall Bricks are controlled and configured by a Security Management Station (SMS), which is a PC running Microsoft Windows-NT. In order to load our rules onto the Lucent VPN Firewall Brick, we had to install our files on the SMS file system, and then cause the SMS to install the new rules on the Lucent VPN Firewall Brick. Since the SMS is a security device, it only accepts rule files that are placed on its file system using its own tools. Therefore, we had to write a short Microsoft Windows-based program, called `lmfload`, which took the *Firmato*-generated rule files and pushed them into the Lucent VPN Firewall Brick using the SMS. We also had to make some mino changes to the output format of the *Firmato*-generated rules so the SMS would be able to display and edit the *Firmato*-generated rules. Maintaining compatibility with the management station was essential in case the administrators needed to make emergency fixes or in case they needed to use a firewall feature that *Firmato* did not support.

After the new firewall was installed and configured using *Firmato*-generated rules, and was running without incident for several weeks, the firewall administration team made several significant changes to the network topology. These included the addition of the fourth interface to the new firewall, and moving several servers into a new DMZ. During these changes, the use of *Firmato* saved the firewall administrators many hours of work, since they did not need to edit multiple rules, and point-and-click on each host-group that needed to change. Instead, a minor change in the topology section of the MDL file was enough to regenerate all the new rule files [Limoncelli, personal communication].

Plans for controlling the access lists on the external router (in front of the firewall) using *Firmato* were in progress as well. The MDL was refined to model the router and its adjacent zones, and additional roles were defined to capture the legacy filtering rules that were implemented by the router's access control lists. However, the field deployment was halted at this point, when the last of this article's authors who was still at Bell Labs (A. Wool), and the senior firewall administrator (T. Limoncelli), left Bell Labs.

9. LESSONS LEARNED AND FUTURE WORK

The *Firmato* project has evolved significantly since it was first designed. Through this evolution, we have learned several valuable lessons about what was right in the *Firmato* model, and what was lacking.

9.1 Successful Features

Firmato's ability to separate the policy from the topology, through the roles, was very important to administrators, though it did require some getting used to. Being able to write a security policy once, and then reuse it at different locations, possibly on different vendors' equipment, was seen as very valuable. Likewise, the higher level of abstraction was seen as a big advantage by users. We believe that these features are crucial for the success of future, large scale, firewall management systems.

Surprisingly, the fact that *Firmato* used, and produced, flat ASCII text files, and did not use a GUI, turned out to be an important feature. Firewall administrators who manage large corporate networks have a great need for scripting and automation. Flat text files are well suited for this type of processing, whereas a GUI-based system requires a lot of time consuming "point-and-click" interaction. Also, flat text files allow administrators to write free-form comments, and to comment-out pieces of the configuration without erasing them.

9.2 Partial Successes

The rule illustrator improved significantly between *Firmato-v1* and *Firmato-v2*, however we believe more can be done in this area. There are two separate issues that need further research.

The first is that on large rule-sets, with more than 20 rules and 20 host-groups, the rule illustration becomes very dense and "spaghetti-like." A step in the right direction was made in the Fang system [Mayer et al. 2000], where we added the ability to view an illustration for a single service at a time. This reduced visual clutter significantly in many cases. We believe the right approach is an interactive display that would let the user zoom in or out, and filter the illustration based on various parameters.

The second, and possibly more difficult issue, is the layout of the host-groups. Users told us that the hierarchical layout growing up for outside host-groups, and down for inside host-groups, is confusing and unnatural. Ideally, the layout should show the network topology, and display the host-group inclusion relationships on top of the topology, with the rules shown over both previous layers. Finding a visually clear and usable mechanism to show all this information is left for future research.

9.3 Limitations and Future Work

With hindsight, we can identify several features that were not incorporated into the *Firmato* design, and limited *Firmato*'s power. Adding these features is left for future work.

(1) *Firmato* was not designed with NAT in mind. This was probably because the Bell Labs network in which we worked did not use any address translation. Unfortunately, the abundant IP address space at Bell Labs is an anomaly, and most organizations make extensive use of NAT. Adding NAT support to *Firmato* is not entirely straightforward: NAT is clearly an aspect of the topology (and not of the roles), yet many vendors associate NAT actions with rules. Also, NAT is often tied closely to routing—which *Firmato* mostly ignores. Finally, NAT violates an implicit assumption in *Firmato*: that an IP address appears in exactly one zone. We have made some initial efforts to model NAT but have yet to design a satisfactory model.

(2) *Firmato* does not let users control the rule order. This is reasonable when rule actions are all "pass" with a trailing "drop" rule. However, users often require finer control. For example, to configure logging, an administrator may wish to write a "redundant" pass rule with logging enabled, and have it appear *before* a more general no-logging pass rule. Likewise, the administrator occasionally wants to exclude a service or a host from a more general group, and it is much easier conceptually (and sometimes requires fewer rules) to implement this negative expressiveness by inserting leading drop rules *before* the pass ones. We believe it is possible to let users influence the rule order while maintaining the role abstractions, by making the rule generation algorithm more sophisticated.

(3) The negative expressiveness of CLOSED roles is too limited, and at the same time somewhat unintuitive. The most natural method for firewall administrators to exclude hosts or services is by "drop" rules. It seems that this can be modeled by introducing "drop" attributes into the entity-relationship model, coupled with a mechanism to influence rule order (as discussed in the previous item).

9.4 Conclusions

We have presented the design and implementation of *Firmato*, a prototype for a new generation of firewall and security management tools. This prototype was used successfully to manage a real-life operational firewall. Thus, we have demonstrated that the task of firewall and security configuration/management can be done successfully at a level of abstraction analogous to modern programming languages, rather than to assembly code. We believe that the central concepts introduced in *Firmato*: separation of policy from topology, abstraction through roles, rule distribution algorithms, configuring multiple devices from a central policy, and multi-vendor support, have been validated by our prototype.

ACKNOWLEDGMENTS

We are grateful to Bill Cheswick, Eric Grosse, Tom Limoncelli, Larry Menten, and Ron Sharp for many useful discussions. We thank Dave Kristol for many helpful comments on an earlier version of this article. Sudip Bhattachariya wrote the IOS back-end and Bruce Wilner wrote the Check Point back-end. We thank the anonymous reviewers for many suggestions that helped us improve the article's presentation.

REFERENCES

AHUJA, R. K., MAGNANTI, T. L., AND ORLIN, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Upper Saddle River, New Jersey.

BARTAL, Y., MAYER, A., NISSIM, K., AND WOOL, A. 1999. *Firmato*: A novel firewall management toolkit. In *Proceedings of the 20th IEEE Symp. on Security and Privacy*. IEEE, Oakland, CA. 17–31.

BELLOVIN, S. M. 1999. Distributed firewalls. *;login: The Magazine of USENIX & SAGE*. 39–47.

CARNEY, M. AND LOE, B. 1998. A comparison of methods for implementing adaptive security policies. In *Proceedings of the 7th USENIX Security Symposium*. Usenix Association, Berkeley. 1–14.

CHAPMAN, D. B. AND ZWICKY, E. D. 1995. *Building Internet Firewalls*. O'Reilly & Associates, Inc.

CHAPMAN, D. W. AND FOX, A. 2001. *Cisco Secure PIX Firewalls*. Cisco Press.

CHESWICK, W. R., BELLOVIN, S. M., AND RUBIN, A. 2003. *Firewalls and Internet Security: Repelling the Wily Hacker*, 2nd ed. Addison-Wesley.

DOT. 2001. Graphviz—open source graph drawing software. version 1.7. `http://www.research.att.com/sw/tools/graphviz/`.

FWB 2002. Firewall builder. `http://www.fwbuilder.org`.

GANSNER, E. R., KOUTSOFIOS, E., NORTH, S. C., AND VO, K.-P. 1993. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng. 19*, 3, 214–230.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

GUTTMAN, J. D. 1997. Filtering postures: Local enforcement for global policies. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, Oakland, CA.

GUTTMAN, J. D. 2001. Security goals: Packet trajectories and strand spaces. In *Foundations of Security Analysis and Design (FOSAD), LNCS 2171*. Springer-Verlag.

GUTTMAN, J. D., HERZOG, A., AND JAVIER THAYER, F. 2000. Authentication and confidentiality via IPsec. In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS), LNCS 1895*. Springer-Verlag.

HELD, G. AND HUNDLEY, K. 1999. *Cisco Access Lists*. McGraw-Hill.

HINRICHS, S. 1999. Policy-based management: Bridging the gap. In *Proceedings of the 15th Annual Computer Security Applications Conference*. Phoenix, AZ.

HLFL 2002. HLFL—high level firewall language. `http://www.hlfl.org`.

HOWE, C. D., ERWIN, B., BARTH, C., AND ELLIOT, S. 1996. What's beyond firewalls? *The Forrester Report 10*, 12 (Nov.).

ICSA LABS. 2003. Certified firewall products. `http://www.icsalabs.com/html/communities/firewalls/certification/rxvendors/index.shtml`.

IOANNIDIS, S., KEROMYTIS, A. D., BELLOVIN, S. M., AND SMITH, J. M. 2000. Implementing a distributed firewall. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*. ACM, Athens, Greece.

LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. 1992. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst. 10*, 4 (Nov.), 265–310.

LIMONCELLI, T. A. 1999. Tricks you can do if your firewall is a bridge. In *First USENIX Conference on Network Administration (NETA)*. USENIX, Santa Clara, CA.

LUCENT 2002. Lucent VPN firewall brick. `http://www.lucent.com/security`.

MAYER, A., WOOL, A., AND ZISKIND, E. 2000. *Fang*: A firewall analysis engine. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, Oakland, CA. 177–187.

REED, D. 2002. Filter language compiler. `http://cheops.anu.edu.au/~avalon/flc.html`.

RUBIN, A., GEER, D., AND RANUM, M. 1997. *Web Security Sourcebook*. Wiley Computer Publishing.

SANDHU, R. S. 1998. Role-based access control. In *Advances in Computers*, M. Zerkowitz, Ed. Vol. 48. Academic Press.

SOLSOFT. 2000. Solsoft NP: Putting security policies into practice. Enterprise Management Associates white paper. `http://www.solsoft.com/library/ema_profiler.pdf`.

Swift, M. M., Hopkins, A., Brundrett, P., Van Dyke, C., Garg, P., Chan, S., Goertzel, M., and Jensenworth, G. 2002. Improving the granularity of access control for Windows 2000. *ACM Trans. Info. Syst. Secu. 5*, 4 (Nov.), 398–437.

Welch-Abernathy, D. D. 2002. *Essential Checkpoint Firewall-1: An Installation, Configuration, and Troubleshooting Guide*. Addison-Wesley.

Wool, A. 2001. Architecting the Lumeta firewall analyzer. In *10th USENIX Security Symposium*. USENIX, Washington, D.C. 85–97.

Wool, A. 2004a. The use and usability of direction-based filtering in firewalls. *Computers & Security 23*, 6, 459–468.

Wool, A. 2004b. A quantitative study of firewall configuration errors. *IEEE Computer 37*, 6, 62–67.

Received February 2003; revised August 2003; accepted January 2004