Firmato: A Novel Firewall Management Toolkit

Yair Bartal* Alain Mayer* Kobbi Nissim† Avishai Wool*

Abstract

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 demonstrate Firmato's capabilities on a realistic example, thus showing that firewall management can be done successfully at an appropriate level of abstraction.*

We implemented our toolkit to work with a commercially available firewall product. We believe that our approach is an important step towards streamlining the process of configuring and managing firewalls, especially in complex, multi-firewall installations.

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 [15]: "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" [11]. A further impediment in this regard is that most available aid (books/advice/tools) is vendor- and case-specific. For instance, [19] is a whole book devoted to configuring the SunScreen Firewall for certain scenarios.

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 configurated 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). 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).[1] In many 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

*Bell Labs, Lucent Technologies, Murray Hill, New Jersey 07974, USA; {yair,alain,yash}@research.bell-labs.com.

†Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel. This author's work was done while visiting Bell Labs, Lucent Technologies; E-mail: kobbi@wisdom.weizmann.ac.il.

[1] 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.

so that the firewall gateway is completely unreachable, 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 and thus the programmer had to know all the idiosyncrasies of the target processor.

These problems get even worse for a medium- or large-sized company, which uses more than a single firewall. These firewalls divide the company's intranets into multiple *zones*, such as human resources, research, etc. 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. And indeed, in each of the few samples of actual corporate firewall configuration files we had access to, we detected "irregularities" with varying degrees of consequences.

1.1. Our Objectives

Our objective is to design and implement a firewall management toolkit, which 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.

1.2. Our Results

Currently, 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. We achieve this by expressing the security policy in terms of *roles*, which we use to define network capabilities. 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 which generates such instances.

3. *A Model Compiler*, which translates a model instance into firewall-specific configuration files. A set of such files typically includes topology and rule-base information. Most modern firewalls allow the filtering capabilities to be distributed onto several gateways, in which case the compiler has to generate a separate set of local configuration files for each gateway. We currently have implemented the compiler with one backend tailored to the *Lucent Managed Firewall (LMF)*.

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.

1.3. Related Work

There are many firewall products on the market, from vendors such as Check Point [4], Cisco [6], Sun Microsystems [19], Lucent Technologies [14], AltaVista, and Network Associates, just to mention a few. (See [9] for an updated list of vendors). Additionally, there are many books on firewall technology and on how to build your own firewall (e.g., [5, 3]). A recent treatment can be found in [1]. While most of the firewall offerings include configuration tools with varying degrees of sophistication, none of these

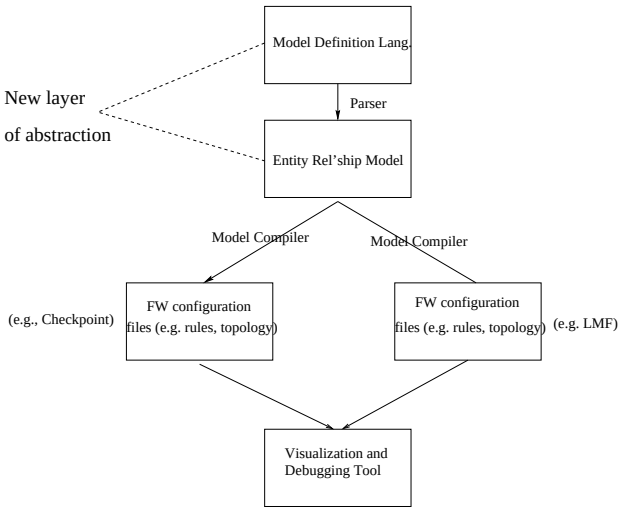

Figure 1. Toolkit Components

vendors seems to focus on firewall and security *management* tools. However, we are currently seeing a few first-generation products being introduced in this arena, e.g., by Check Point, Cisco, and SOLsoft [7].

The work closest in spirit to ours is probably Guttman's work on filtering postures [10]. 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). While an important step towards security management, [10] 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 re-use much harder. Furthermore, while the specification is firewall independent, it (and the localization method) does not seem to lend itself easily to be used with actual firewalls.

Role-based access control (RBAC) (see [16] for a survey) and research in trusted/secure operating systems with adaptive security policies (see, e.g., [2]) 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 and with topology-related issues. 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 [13]; there a principal can act in a role to express acting as a different, weaker principal.

Organization: In Section 2 we introduce in some detail our entity relationship model and the model definition language MDL. Section 3 covers the design of our model compiler. In Section 4 we discuss the rule illustrator. We then walk the reader through a complete realistic example of using our toolkit in Section 5. We mention possible extensions in Section 6, and we conclude with Section 7.

2. The Modeling Framework

In this section we present our modeling framework. We start with a brief definition of the terminology we use. Then in Sections 2.2 and 2.3 we introduce our modeling concepts, and the details of our entity-relationship model. In Section 2.4 we describe our model definition language MDL.

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 machines, which can be either firewalls or routers.

Interfaces: By definition a gateway is *multi-homed* (see [18]), since it has at least two Internet connections. Each connection goes through an *interface*, which has its own unique IP address. 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).

Zones: The gateways partition the IP address space into disjoint *zones*. 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.

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.

2.2. Modeling Concepts

The bulk of our modeling framework captures the topology-independent part of the security policy. The main concept we introduce is that of a *role*. A role is a property that may be assumed by different hosts in the network. Roles define capabilities of initiating and accepting services. For example, the role of a `mail_server` might define the capability of accepting mail service (`smtp`, i.e., `tcp` at port 25) from anywhere. A slightly more sophisticated policy might introduce the roles `mail_server` and `internal_mail_server`. The role of the internal mail server defines the capability of accepting `smtp` from peers

assuming the role of `mail_server`. Mail servers with a robust "sendmail" module (typically located in a "demilitarized zone", or DMZ) assume the role of `mail_server`, whereas regular mail servers (inside the intranet) assume the role of `internal_mail_server`.

We allow roles to specify capabilities not only of accepting, but also of initiating. This is not strictly necessary, since every service has a accepting side. However, we argue that it is natural to define, for example, an outbound role such as `web-enabled`, which allows `http` to the outside world. Such a role is typically attached to intranet hosts, allowing employees to browse the web.

From these simple examples we see that a role defines essentially (1) the allowed service(s) and (2) the peers to or from which the services apply. The peers are expressed in terms of other (or the same) role(s). Eventually, roles are assigned to actual hosts (machines), thus binding the policy (roles) to the actual topology. For convenience we also allow *role-groups* to be defined and assigned to hosts—these are simply collections of roles.

Roles denote *positive* capabilities, and implicitly realize a "whatever is not explicitly allowed is disallowed" strategy, e.g., a host accepts an `http`-request if and only if it was assigned a corresponding role of `web-server`.

Like the security policy through the role abstraction, the network topology is modeled also by an entity-relationship approach. A *host* entity models a machine on the network with its own IP address and name. A *host-group* represents a set of machines, either defined via an IP address range or a set of other host(-group)s. Roles can be attached to both host and host-group entities. We make use of *inheritance* in the natural way, such that the set of roles assumed by a host h is the set of all roles assigned to some host-group that includes h.[2]

Consider now the example of a gateway, connecting a subnet for payroll with the rest of the intranet. It is important to ensure that access to the gateway be limited to a few administrative tasks from a small set of other machines; otherwise routing and access control could be easily corrupted. In other words, the gateway should be "stealthed". While it is easy to assign roles to the gateway which only allow very limited capabilities, such as remote administration from hosts with a `fw_admin` role, the gateway might inherit other roles, allowing undesirable access. To curb this possibility, we introduce the notion of a *CLOSED* group of roles:

Definition 1 *A CLOSED role-group is a role-group for which inheritance of roles does not apply: a host h which assumes a CLOSED role-group does* not *inherit other roles assigned to any host-group A which contains h. A host may assume at most one CLOSED role-group. Role-groups which are not CLOSED are called OPEN. By default role-groups are OPEN.*

[2]Host-group h_1 includes host-group h_2 if the IP address range of h_2 falls completely within the IP address range of h_1.

Recall that roles express *positive* capabilities. The "*CLOSED*" mechanism offers a limited form of negative expressiveness. As an aside, we note that roles alone (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. The design issue thus becomes how to allow a simple, yet adequate, form of negative expressiveness to the security administrator. We believe that the "*CLOSED*" mechanism captures just enough expressiveness. As illustrated in the example above, it allows *group exclusion*, which we found to be the most prevalent form of negative expression in security policy design.

2.3. Our Entity-Relationship Model

We now describe our model framework in some more detail. See Figure 2 for a pictorial representation.

A `Role` entity consists of a set of `Peer-Capabilities`. Each such `Capability` 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 capability, 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 which are assigned a *CLOSED* role-group do not inherit other roles.

We model the network topology as follows: the network is partitioned into `Zone`s, connected through `Gateway`s. A `Gateway` consists of a `Gateway-Interface` for each adjacent `Zone`. Each `GatewayInterface` has its own IP-address (modeled by the `Interface-Host` attribute). 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`) and

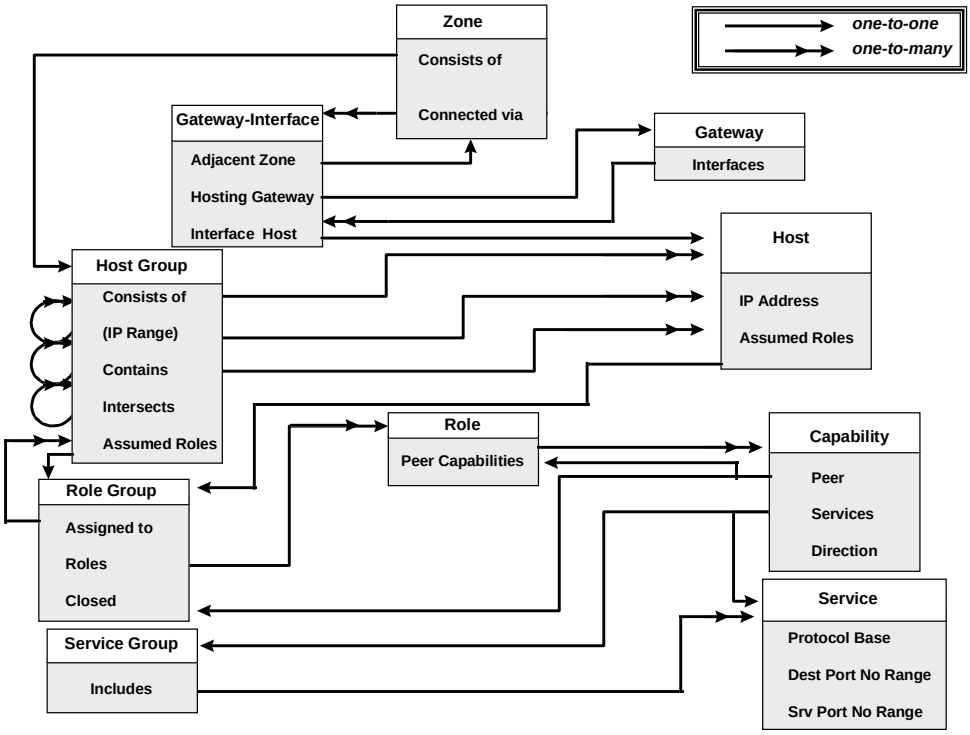

Figure 2. The Entity-Relationship Model

thus this is the place where the policy (modeled by roles and role-groups) is linked to the network topology.

Remark: Our model framework is easily extensible. As firewall capabilities evolve, new attributes can be added to objects via inheritance, or whole new objects can be added *without* invalidating the original model. Examples of such advanced firewall features are (1) time-dependent roles, whose capabilities may depend on the time of the day, and (2) session interdependence, where a role's capability depends on the current traffic pattern. More model extensions are discussed in later sections.

2.4. 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. We envision that a further development of our system will include a graphical user interface (GUI), assisting or even replacing the MDL for the design and specification of a model instance. Here we give a definition-by-example of MDL; an example with a complete MDL program can be found in Section 5.

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

2.4.1 MDL for Security Policy Description

A `Service` is defined via a statement of the form

```
<service-name> '='
   <protocol-base>
  '['<Dest-Port-No-Range> ','
  <Src-Port-No-Range>']'
```

The following code fragment defines the widely used services `smtp`, `ssh`, `ping`, `https` and a service denoting all `tcp`-based packets:

```
SERVICES {
  smtp   = TCP [25]
  ssh    = TCP [22]
  ping   = ICMP [8,0]
  https  = TCP [443]
  all_tcp   = TCP [*]
}
```

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

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

The following code fragment defines the two service-groups `admin-to-gtwy, gtwy-to-admin`:

```
SERVICE_GROUPS {
  admin_to_gtwy = {ssh, ping}
  gtwy_to_admin = {ssh, https}
}
```

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

The following code fragment defines roles `mail_server` and `internal_mail_server` we discussed in Section 2.2. The roles `gateway_in` and `gateway_out` model the capabilities of gateway interfaces in each direction. This example is continued in the next code fragment.

```
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 which assume this role-group.

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

2.4.2 MDL for Topology Description and Policy Mapping

Hosts and host-groups are defined by the following statements:

```
   <host-name> '=' '[' <IP-Addr>
     ']' ':'  <role-grp-name>
   <host-grp-name> '=' '['
     <IP-Range> ']' ':'
     <role-grp-name>
```

The following code fragment defines the hosts *dirty* (presumably outside the intranet), *dusty*, assigning them roles of external and internal mail servers, respectively.

```
HOST {
 dirty = [ 111.222.100.6 ]
   : mail_server
 dusty = [ 111.222.1.3 ]
   : internal_mail_server
}
```

Gateways are defined by the following statement:

```
   <gateway-name> =
   '{' <host-name1> ',' <host-name2>
     ...  '}'
```

The following code fragment defines `payroll_gw_interface1/2` as hosts and specifies their IP-addresses, and then defines the gateway `payroll_gw` as having `payroll_gw_interface1/2` as its two interfaces. It also assigns the role-group `gateway` to the interfaces.

```
HOST {
 payroll_gw_interface1 =
      [ 111.222.26.226 ] : gateway
 payroll_gw_interface2 =
      [ 111.222.24.210 ] : gateway
}

GATEWAYS {
 payroll_gw  = {payroll_gw_interface1,
                payroll_gw_interface2}
}
```

Zones are defined via the following statement:

```
   <zone-name> ':'  '{'
     <gtwy-interface-name1> ','
     <gtwy-interface-name2> ...'}'
```

The following code fragment first defines the zones `payroll_zone` and `corp_zone` (parts of the intranet `manhattan_office`) as host-groups, specifies their IP-Ranges. It then defines parts of the network topology

by specifying the `payroll_zone` to be connected to the `payroll_gw` via the `payroll_gw_interface1`, and the second interface of the `payroll_gw` to be connected to the `corp_zone`.

```
HOST_GROUPS {
 manhattan_office =
    [111.222.0.0-111.222.255.255]
     : intranet_machine
 payroll_zone =
    [111.222.26.0-111.222.26.255]
     : payroll_machine
 corp_zone =
    [111.222.24.0-111.222.24.255]
     : non_payroll_machine
}

ZONES {
 payroll_zone : {payroll_gw_interface1}
 corp_zone : {payroll_gw_interface2, ...}
}
```

3. The Model Compiler

After the security administrator has designed the security policy and programmed it in MDL and the MDL parser has generated the entity-relationship model, this model 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.

3.1. A Generic Rule Format

It is straightforward to generate the services and host-group configuration files; thus we omit the details. For the rest of this section, we focus on the interesting part of the compiler, which deals with rule-base generation. We do so in terms of a 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. We shall say more about the use of this field in Section 3.3.

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.

3.2. The Basic Model Compiler

The generation of the rule-bases can be separated into two parts: first creating a basic centralized rule-base, and then adapting the rule-base to each of the gateway interfaces. We cover these tasks in turn.

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-base topology adaptor.

Role definitions are a description of what operations are allowed between machines according to their assumed roles. 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 and 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 which 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 which 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;
   insert the rules into the rule-base
```

Phase 2:

```
foreach pair of a CLOSED host-group H1
        and an OPEN host-group H2:
   generate all the positive rules
      between H1 and H2;
   foreach CLOSED host-group G
        contained in H2:
      generate negative rules
           between H1 and G;
      insert the negative rules
           into the rule-base;
   insert 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 ''*''
      and insert it into the rule-base
           /* the normal case */
foreach pair of OPEN host-groups:
   generate all the positive rules between them
      and insert 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 H1:
   foreach statement of the form {r $ R : s}:
        /* R=role-group; $=direction;
           s=service; */
      if H2 is CLOSED:
        /* create a rule if H2 has role r */
        if the role-group assigned to H2
             contains a role in R
           create a positive rule between
             H1 and H2 with service=s
      otherwise for all host-groups G
           that contain H2:
        if the role-group assigned to G
             contains a role in R
           create positive rule between
             H1 and H2 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 case analysis for the general case assures that our algorithm generates a rule-base which correctly implements the security policy defined by the model. We omit the details.

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. As an aside, we note that firewalls themselves can perform internal optimizations to increase the filtering speed. By appropriate pre-processing, the firewall can test packets at a rate that is essentially *independent* of the number of rules in the rule-base (see [17, 12] for further details). Thus the actual number of rules is not the only factor in determining the resulting firewall performance.

3.3. The Rule-Base Topology Adaptor

Thus centralized rule-base, generated by the basic model compiler, needs to be distributed to each of the gateways in the network with appropriate customization for each interface. To ensure the security policy is observed we must include all rules that concern a certain pair of hosts in all gateways along any possible routing path between them. In order to avoid making assumptions on the routing protocols and to add tolerance to routing failures, a "safe" strategy is one that replicates the centralized rule-base onto every interface on every gateway. This is basically the strategy we adopt.

We do, however, need to set the direction field of the rules. This is important in reducing the possibility of spoofing attacks. The way the direction field is used by the firewall is as follows. The firewall checks the direction in which the packet enters the gateway interface and compares it with the direction field of the rule. If the packet is attempting to leave the interface into the adjacent zone, it is allowed only if the rule's direction is IN or BOTH. Likewise, a packet is allowed to enter the interface from the adjacent zone only if rule's direction is OUT or BOTH.

We reiterate that the direction field is *not* implied by the role's `Direction` attribute—which is captured by the fact that one host group is designated the *source* while the other is the *destination*.

In the absence of other information, we can always set the direction fields to BOTH. However, we would like to set them as precisely as possible (i.e., to decrease the number of rules with a BOTH direction), since this allows the firewalls to ensure that packets which claim to arrive from a host h indeed appear only on the gateway interface that leads to h.

In a general network topology, if no assumptions are made on routing protocols, the source-destination pair does not imply much information about the direction of the packet, except if the source or the destination resides in the adjacent zone to the gateway interface. This leads to the following algorithm for our topology adaptor:

```
Replicate the centralized rule-base
      to every gateway interface
   foreach gateway interface:
      foreach rule in the rule-base:
         if the source is in the
               adjacent zone
            set direction=OUT
         else if the destination is
               in the adjacent zone
            set direction=IN
         else set direction=BOTH
```

We emphasize that this is only the first step towards the desirable goal, and the algorithm can be significantly improved for many common cases. In order to improve on the above algorithm (in terms of avoiding unnecessary replication and spoofing prevention), some knowledge about routing assurances must be available. This knowledge could be placed in a rule optimizer or could be made part of an extension to the entity-relationship model.

4. 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-groups 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. The actual graph layout was produced by the daVinci V2.1 visualization system [8]. An example of the illustrator's output appears in Figure 4, visualizing the rules of the configuration we discuss in Section 5.

4.1. 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. A dashed black edge indicates host-groups that intersect (i.e., 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.[3] One category is called the "outside", typically the one that contains the Internet zone, while 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 diamond-shaped nodes in the middle of the graph (other host-groups are shown as ovals). Then we display the "inside" host-groups as a tree that grows downwards from the `_in` node, and the "outside" ones as a tree growing upwards 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.

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

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

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.

4.2. Adding the Rules

The illustrator displays the rules only for services that cross the interface—other rules (dealing with services where both endpoints 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, e.g., `all_tcp` is a red arrow, `telnet` is a blue arrow, etc.

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 capabilities of other host-groups that contain it. Thus ignoring these negative rules has minimal effect on the meaning of the graph.

5. 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).

5.1. The Environment

The external firewall guards the corporation's Internet connection. 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 [18]. The corporation actually only uses two hosts to provide these services, one for `dns` and the other (called `multi_server`) for all the other services.

Behind the DMZ is the internal firewall 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. Securing the firewall administration host is critical to the network's integrity, so it is deemed appropriate to separate it from the other corporate hosts.

Within the corporate network zone, there is one distinguished host, `control`, which provides the administration for the servers in the DMZ.

5.2. Policy Goals

The policy we consider is a rather simple one, which nonetheless covers many of the aspects which 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 made public explicitly. 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` 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).

5.3. 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`, etc. In the interests of brevity, we omit the straight-forward 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.[4] The role `R_none` is a role that does not have *any* privileges; nevertheless it is a role like any other, and needs to be declared. The meaning of the other roles is clear from their names.

```
ROLES {
 R_none
 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
}
```

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

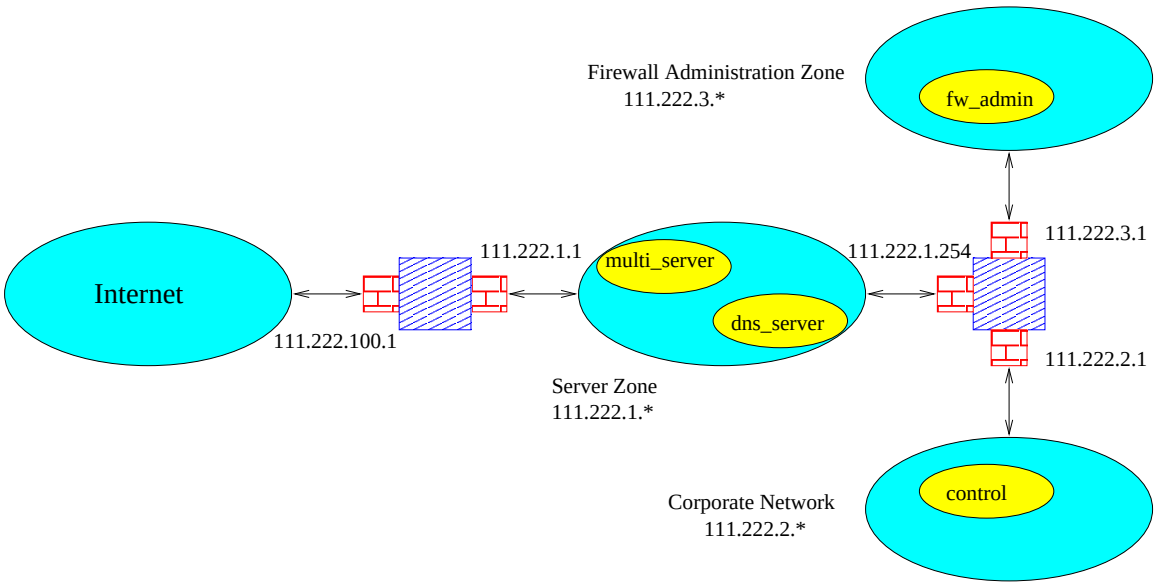

Figure 3. The example network topology

Next we define two role-groups. The first (`R_multi_server`) is a container group for the various roles that are played by the `multi_server` host. The second group, `R_gw` is more interesting: 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_multi_server = {R_mail_server,
          R_web_server, R_ftp_server}
 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.

```
ROLE_DEFINITIONS {
 R_admin -> R_interface : admin_to_gtwy
 R_admin <- R_interface : 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_multi_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 used the network topology in a very limited way. Clearly, we had the network topology in the back of our mind when we defined the various roles, e.g., when we defined a role-group for the three roles played by the `multi_server` host. How-

ever, the definitions themselves are not tied to particular IP addresses, or even to the broader network structure we have. Thus essentially 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` in the definition

  ```
  R_corporate -> * : all_tcp
  ```

  as a side effect we would have inadvertently given hosts with the role `R_corporate` the capability to update the DMZ servers, since "*" 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*, e.g., hosts in this group reside on both sides of the external gateway. We have seen a few real examples where such zone-spanning host-groups could have allowed spoofing attacks[5]. This is since the gateways were configured to allow a service to such host-groups in both directions and thus were blind to the spoofing. 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.

5.4. 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 host-groups for the four zones. We choose not to give the server zone or the firewall administration zone any roles (using the `R_none` role) since the machines inside each of these zones play different roles. Nevertheless we must define the host-groups since the are required in the zones definitions which appear below.

```
HOST_GROUPS {
Z_dmz    = [111.222.1.0-111.222.1.255]
      : R_none
Z_corp   = [111.222.2.0-111.222.2.255]
      : R_corporate
Z_admin  = [111.222.3.0-111.222.3.255]
      : R_none
Z_internet = [0.0.0.0-111.221.255.255,
          111.223.0.0-255.255.255.255]
      : R_internet
```

Next we define the five gateway interfaces, and assign each of them the role `R_gw`. 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 need to give each interface a separate definition (rather than grouping them together into a single host-group) since we need to refer to them individually in the gateway and zone definitions which follow.

```
# The gateway interfaces.
I_internet_dmz = [111.222.100.1] : R_gw
I_dmz_in       = [111.222.1.1]   : R_gw
I_dmz_corp     = [111.222.1.254] : R_gw
I_corp_in      = [111.222.2.1]   : R_gw
I_admin        = [111.222.3.1]   : R_gw
```

The last host-groups we define are the various special machines in the network: the DMZ servers `multi_server` and `dns_server`, the firewall administration machine `fw_admin`, and the web administration machine `control`.

```
multi_server   = [111.222.1.17]
    : R_multi_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
}
```

We complete the topology-specific definitions by defining the gateways with the interfaces attached to them, and the zones with the interfaces that form their border.

```
GATEWAYS {
dmz_gw  =
   {I_internet_dmz, I_dmz_in}
corp_gw =
   {I_dmz_corp, I_corp_in, I_admin}
}

ZONES {
Z_internet : { I_internet_dmz }
Z_dmz      : { I_dmz_in, I_dmz_corp }
Z_corp     : { I_corp_in }
Z_admin    : { I_admin }
}
```

[5] We are not aware of such attacks actually occurring. Nevertheless the firewall configurations were potentially vulnerable.

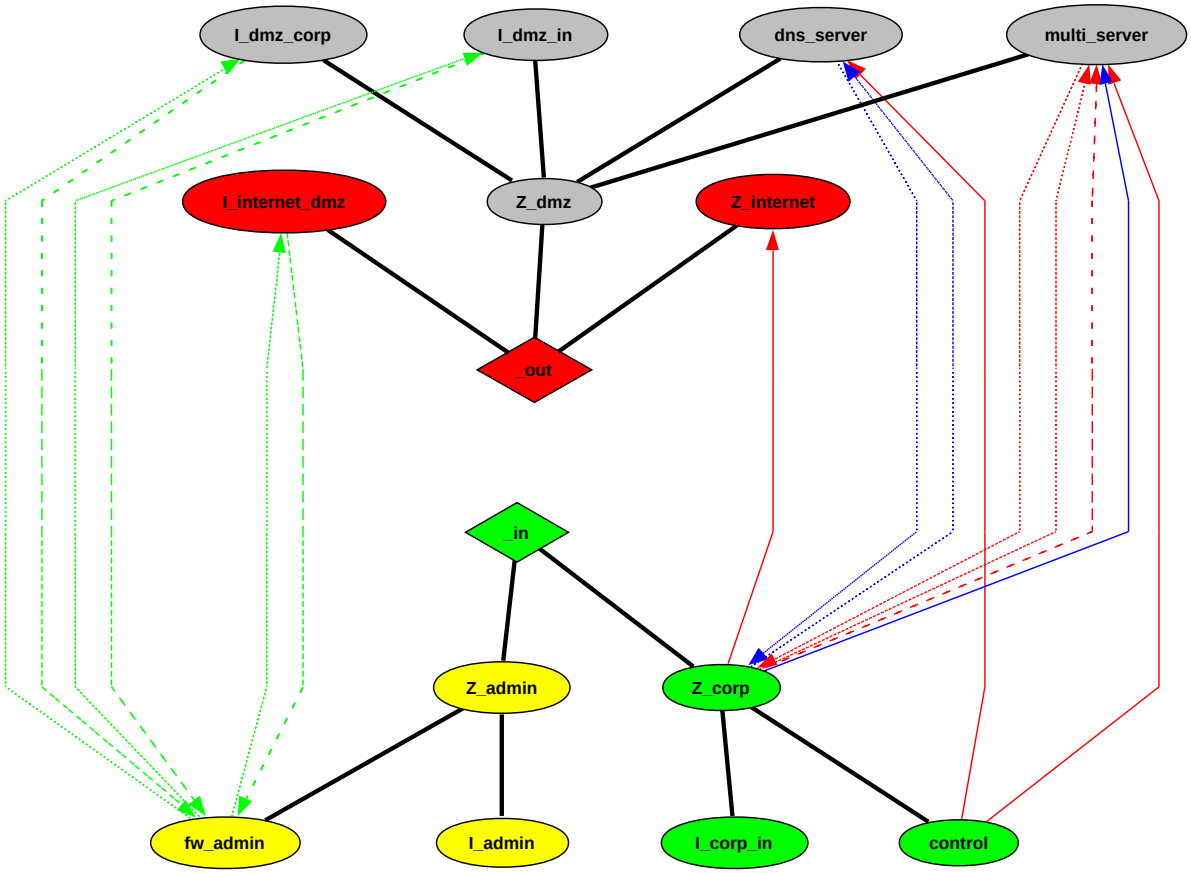

Figure 4. The output of the Rule Illustrator, from the point of view of the `I_dmz_corp` interface.

5.5. Compiling and Visualizing the Rules

We ran the *Firmato* compiler on the example files we discussed in Sections 5.3–5.4. The compiler generated 42 rules, 10 of which were negative due to the *CLOSED* `R_gw` role group. A sample of the generated rules is:

```
(rule_number=1,
   src_host=fw_admin,
   dst_host=I_internet_dmz,
   service=admin_to_gtwy,
   action=pass, direction=BOTH)
(rule_number=12,
   src_host=*,
   dst_host=I_internet_dmz,
   service=*,
   action=drop, direction=BOTH)
(rule_number=21,
   src_host=Z_corp,
   dst_host=Z_internet,
   service=all_tcp,
   action=pass, direction=BOTH)
```

Figure 4 shows a graphic representation of the generated rules from the point of view of the `I_dmz_corp` interface, that separates the corporate network and the firewall administration zones from the server zone and the Internet (recall Figure 3).

As we discussed in Section 4, the figure separates the host-groups into "inside" and "outside" and shows the inclusion relations between host-groups. E.g., host-group `_in` 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 line pattern. E.g., the edge from `Z_corp` to `Z_Internet` corresponds to the `all_tcp` service.

6. Model Extensions and Future Work

As mentioned in Section 2, many firewalls have advanced features which are not covered by our basic model, such as time-dependency (e.g., a rule applies only at certain times of the day) and session interdependency (e.g., allowing an incoming RealAudio stream actually means allowing an incoming `udp` session only if there is a corresponding outgoing `tcp` session). The design of our model is such that through inheritance and additions we can extend the model without invalidating the existing part.

Our focus in this paper has been on packet-filtering firewalls, but we believe that our modeling efforts (Section 2) should apply also to application proxy filtering, albeit with a somewhat different back-end for the compiler.

A more substantial extension is the design and implementation of a *policy monitoring tool*. We can extend the basic model with entities and relationships modeling actual sessions passed or dropped at the various gateways. The firewall can then potentially (through its log data) *instrument* this new part of the model. Consequently, we can compare the set of "current sessions" with the security policy to verify that indeed each host is only involved in sessions corresponding to one of its roles. The policy monitoring tool can potentially also be used as a basis for intrusion detection tools. Another direction for extending our model is for *virtual private network* (VPN) management. There, roles for encryption gateways, authenticators, key distributors, etc. can be designed and added to our model.

Other kinds of extensions include a graphical user interface (GUI) for the MDL (or to replace the MDL). Future work in a more foundational direction includes a formal verification of our compiler algorithm.

We have recently optimized the Topology Adaptor of Section 3.3 to minimize the replication of rules onto the various interfaces. This modification both improves the anti-spoofing robustness of our system, and shortens the resulting rule-bases.

7. Conclusion

We have presented an initial design and implementation of *Firmato*, a prototype for a new generation of firewall and security management tools. We have shown its usefulness on a real world example, demonstrating 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 assembly code.

In a more general context, we view *Firmato* as an important first step towards the convergence of security and network management.

Acknowledgments

We are grateful to Eric Grosse, Binay Sugla, Danny Raz, P. Krishnan, and Ron Sharp for many useful discussions. Many thanks to Amy Axelrod for her help with the illustrations. We thank Dave Kristol and the anonymous referees for many helpful comments on an earlier version of this paper.

References

[1] J. P. Anderson, S. Brand, L. Gong, T. Haigh, S. Lipner, T. Lunt, R. Nelson, W. Neugent, H. Orman, M. Ranum, R. Schell, and E. Spafford. Firewalls: An expert roundtable. *IEEE Software*, 14(5):60–66, Sept./Oct. 1997.

[2] M. Carney and B. Loe. A comparison of methods for implementing adaptive security policies. In *Proceedings of the 7th USENIX Security Symposium (SECURITY-98)*, pages 1–14, Berkeley, Jan.26–29 1998. Usenix Association.

[3] D. B. Chapman and E. D. Zwicky. *Building Internet Firewalls*. O'Reilly & Associates, Inc., 1995.

[4] Check Point FireWall-1, version 3.0. White paper, June 1997. `http://www.checkpoint.com/products/whitepapers/wp30.pdf`.

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

[6] Cisco's PIX firewall series and stateful firewall security. White paper, 1997. `http://www.cisco.com/warp/public/751/pix/nat_wp.pdf`.

[7] A. Fremont. Net Partitioner 3.1. SOLsoft white paper, 1998. `http://www.solsoft.com/np/whitepapernplet.pdf`.

[8] M. Fröhlich and M. Werner. The graph visualization system daVinci, version 2.1. `http://www.informatik.uni-bremen.de/~davinci/`, July 1998.

[9] C. Fulmer. Firewall product overview. `http://www.waterw.com/~manowar/vendor.html`, Oct. 1998.

[10] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. IEEE Symp. on Security and Privacy*, Oakland, CA, 1997.

[11] C. D. Howe, B. Erwin, C. Barth, and S. Elliot. What's beyond firewalls? *The Forrester Report*, 10(12), Nov. 1996.

[12] T. V. Lakshman and D. Stiliadis. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, 1998.

[13] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. Computer Systems*, 10(4):265–310, Nov. 1992.

[14] Lucent managed firewall, version 2.0. White paper, 1998. `http://www.lucent.com/iss/pdf_download/lmf_technical.pdf`.

[15] A. Rubin, D. Geer, and M. Ranum. *Web Security Sourcebook*. Wiley Computer Publishing, 1997.

[16] R. S. Sandhu. Role-based access control. In M. Zerkowitz, editor, *Advances in Computers*, volume 48. Academic Press, 1998.

[17] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast scalable algorithms for level four switching. In *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, 1998.

[18] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[19] K. M. Walker and L. Croswhite Cavanaugh. *Computer Security Policies and SunScreen Firewalls*. Sun Microsystems Press, 1998.