Uma Abordagem Baseada em Programação Declarativa para Configuração de Firewalls em Ambientes Heterogêneos

Cássio Ditzel Kropiwiec[1], Edgard Jamhour[2], Mauro Sérgio Pereira Fonseca[2], Fabrício Enembreck[2], Guy Pujolle[1]

[1]Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie Paris – France

[2]Programa de Pós-Graduação em Informática Aplicada
Pontifícia Universidade Católica do Paraná (PUCPR) – Curitiba, PR – Brazil

{cassio, mauro, jamhour, fabricio}@ppgia.pucpr.br, guy.pujolle@lip6.fr

Abstract. *This paper presents a framework for high-level security policy representation for firewall configuration, independent of topology, devices and firewall capabilities, based on declarative programming. The developed algorithm indicates the rules that must be applied to each firewall, modifying the rules according to the firewall capabilities if necessary. The algorithm includes a process that evaluates the generated rules to certify that they don't violate the security. A case study is presented to demonstrate the effectiveness of the framework.*

Resumo. *Este artigo apresenta um framework para representação e tratamento de políticas de segurança de alto-nível para firewalls, independente de topologia, dispositivos e características dos dispositivos, baseada em programação declarativa. O algoritmo desenvolvido é capaz de indicar as regras que devem ser aplicadas a cada firewall, modificando-as de acordo com as capacidades dos firewalls. O algoritmo inclui um processo de avaliação das regras geradas para verificar se a política não é violada, e, em caso de violação, indicar a causa do problema. Um estudo de caso é apresentado para comprovar o funcionamento do framework.*

1. Introdução

As redes de computadores estão presentes em todos os lugares, desde pequenas escolas até grandes corporações. Redes pequenas normalmente possuem apenas um equipamento do tipo roteador ou *firewall*, para interligá-las à Internet. Por outro lado, em redes maiores, é comum a existência de vários *firewalls*, com o objetivo de dividir essa rede em subredes menores, simplificar o gerenciamento e aumentar a segurança da rede. Essa divisão também possibilita que o administrador especifique regras para permitir ou bloquear o acesso de certos usuários internos aos recursos existentes na própria rede, já que muitos ataques são originados a partir da rede interna [Markham e Payne 2001].

Um problema que o administrador enfrenta na configuração de uma rede de grandes proporções é determinar o conjunto de regras que deve ser aplicado a cada *firewall*, de forma que a segurança seja corretamente atendida, sem permitir falhas de

segurança ou redundância de configuração [Al-Shaer e Hamed 2004]. Esse problema também ocorre na inclusão ou exclusão de regras, pois o administrador pode ser obrigado a verificar todos os *firewalls* para certificar-se que não houve comprometimento da segurança.

Outro problema que pode ocorrer é a rede ser heterogênea, possuindo *firewalls* de diversos modelos e diversos fabricantes, com recursos distintos uns dos outros. Com isso, além de avaliar a distribuição das regras, o administrador deve levar em consideração os recursos para determinar se os *firewalls* podem ou não atender às regras.

O objetivo deste trabalho é descrever um *framework* composto por uma linguagem baseada em programação declarativa para representação de políticas de alto nível, independente de topologia e de dispositivos, e um algoritmo para determinar a distribuição das regras para cada *firewall* presente na rede. Esse mecanismo é capaz de informar ao administrador caso existam regras que não podem ser satisfeitas sem comprometer a segurança, devido a limitações dos *firewalls*.

Esse artigo é organizado da seguinte forma: a seção 2 apresenta os trabalhos relacionados. A seção 3 descreve a motivação para o desenvolvimento deste trabalho. A seção 4 detalha a proposta, apresentando a linguagem e o algoritmo desenvolvidos. A seção 5 apresenta um estudo de caso, para demonstrar o funcionamento do *framework*. Por fim, a seção 6 conclui o artigo e apresenta os trabalhos futuros.

2. Trabalhos Relacionados

Várias linguagens se propõem a representar as regras de configuração de *firewalls*. Entretanto, várias delas focam aspectos específicos da configuração de segurança, de forma que nem sempre podem ser aplicadas a todos os casos.

Para facilitar o entendimento, as linguagens podem ser classificadas de acordo com suas características: dependentes ou não de dispositivo, dependentes ou não de topologia e baseada em regras ou baseada em políticas.

Com relação à dependência de dispositivo, existem linguagens que são específicas para determinados *firewalls*, normalmente criadas pelos próprios fornecedores. Entre elas podem-se citar as linguagens Cisco PIX [Cisco 2004a] e Cisco IOS [Cisco 2004b]. Essas linguagens possuem a limitação de não poderem ser utilizadas para a representação da configuração em redes com *firewalls* de diferentes fabricantes, já que cada um possui sua própria linguagem. No caso das linguagens independentes de dispositivo, existe a vantagem de não serem limitadas ao *firewall* do próprio fabricante. Esse é o caso da linguagem INSPECT [CheckPoint 2005] da CheckPoint. É uma linguagem que pode ser compilada e aplicada a diferentes *firewalls*, mas que, entretanto, requer que esses equipamentos possuam os requisitos mínimos exigidos.

Com relação à dependência de topologia, as linguagens podem ser dependentes, quando foram projetadas para representar a configuração de cada *firewall*, considerando que as redes ou dispositivos a ele ligados não se modificam ao longo do tempo, ou independentes de topologia, quando é possível representar a política global independente da representação da topologia. No segundo caso, normalmente não são indicadas as regras que serão aplicadas em cada equipamentos. Um mecanismo ou

algoritmo é encarregado de avaliar a topologia e traduzir a configuração global para as regras de cada equipamento. Dessa forma, é possível modificar a topologia sem haver necessidade de alterações na política.

Alguns exemplos de linguagens dependentes de topologia podem ser encontrados em [Lee, Yusuf, Luk, Sloman, Lupu, e Dulay 2003] e [ROPE 2005]. O trabalho [Lee, Yusuf, Luk, Sloman, Lupu, e Dulay 2003] descreve um *framework* para representar a configuração de um *firewall* como descrições de alto nível baseadas na linguagem de especificação Ponder [Damianou, Dulay, et al. 2001]. A descrição de alto nível consiste de duas partes: a especificação de controle para cada *firewall* individualmente e as hierarquias de endereços IP e serviços. Para representar a configuração de uma rede com vários *firewalls*, é necessário criar uma especificação de controle para cada um deles, tornando essa especificação dependente da topologia. Já a linguagem descrita em [ROPE 2005] é uma linguagem de *scripts*, sendo que o objetivo é tornar a representação da configuração do IpTables (configuração do *firewall* do Linux) mais robusta. Por ser uma extensão do IpTables, mantém suas características, sendo necessário indicar a configuração de cada *firewall* existente na rede.

No caso das linguagens independentes de topologia, é possível classificá-los como baseadas em políticas e baseadas em regras. Nas linguagens baseadas em regras, a configuração da rede é representada por um conjunto de regras do tipo condição-ação, onde a condição indica em que situação a regra deve ser aplicada, e a ação determina a operação a ser realizada. Em alguns casos, é permitido realizar agrupamento dos endereços e serviços. Alguns exemplos de linguagens baseadas em regras podem ser encontrados em [DeTreville 2002], [Haixin, Jianping e Xing 2000].

DeTreville [DeTreville 2002] propõe uma linguagem de segurança baseada em lógica chamada Binder, para expressar políticas e autorizações em sistemas distribuídos abertos. O Binder é uma extensão da linguagem de programação lógica Datalog, que, por sua vez, é um subconjunto do Prolog puro. Apesar do pequeno número de primitivas, o Binder é apresentado como sendo mais expressivo que ouras linguagens de segurança existentes. Além disso, o Binder utiliza o conceito de contexto. Cada componente em um ambiente distribuído possui seu próprio contexto local. Extensões ao Datalog permitem que um contexto remoto (assinado por um certificado) possa ser utilizado em declarações locais. A primitiva utilizada para representação de contextos remotos é "*says*", com a semântica "algo é verdade se houver um certificado atesta isso".

Em [Haixin, Jianping e Xing 2000] é descrito um *framework* em que é possível representar a política de segurança da toda a rede de forma centralizada, e define um algoritmo para distribuição das regras pelos *firewalls*. Entretanto, não é definida uma abstração para os endereços de origem e destino, representando diretamente os endereços IPs e a ação a ser executada. Isso faz com que esse trabalho não seja totalmente independente de topologia, pois é possível modificar os *firewalls* e redes existentes entre a origem e o destino, mas não se pode mudar os IPs de origem e destino sem modificar as regras da política.

Por outro lado, as linguagens baseadas em políticas utilizam um conceito mais abstrato em que políticas de segurança descrevem "o que deve ser feito" ao invés de "como deve ser feito", ou seja, a política define a intenção independentemente dos

mecanismos utilizados para implementá-la. Exemplos de linguagens baseados em políticas podem ser encontrados em [Ou, Govindavajhala e Appel 2004], [Burns, Cheng, Gurung, Rajagopalan, Rao, Rosenbluth, Surendran e Martin Jr. 2001] e [Guttman 1997].

Em [Ou, Govindavajhala e Appel 2004] é apresentado um *framework* composto por uma linguagem de alto nível para representação de políticas e pelos algoritmos para verificar se as configurações de baixo nível correspondem à política de alto nível. As regras de alto nível especificam uma operação (de leitura ou escrita) que um usuário pode executar sobre um objeto (ou dado), e são sempre positivas, ou seja, não existem regras de negação de acesso (tudo o que não é explicitamente permitido é proibido). Esse trabalho permite, além das configurações de *firewall*, indicar configurações de segurança para computadores e aplicações. Por esse motivo, é necessário que esses computadores possuam um mecanismo que permita que a política seja efetivamente aplicada.

Por sua vez, em [Burns, Cheng, Gurung, Rajagopalan, Rao, Rosenbluth, Surendran e Martin Jr. 2001] é apresentado um projeto para gerenciamento automático de políticas de segurança em redes dinâmicas. O componente central é um motor de políticas com modelos de elementos de rede e serviços que valida as políticas e gera novas configurações de segurança para os elementos de rede quando a segurança é violada.

[Guttman 1997] introduz uma linguagem para representar políticas de controle de acesso para redes do tipo que os filtros de pacotes podem implementar. A linguagem permite abstrair o endereço das redes e dos computadores, através da utilização de nomes. Junto com a linguagem, define um algoritmo que, para uma dada topologia, é capaz de criar o conjunto de filtros para cada um dos *firewalls* ou roteadores da rede; e um segundo algoritmo para verificar se o conjunto resultante de filtros viola alguma das políticas de acesso da rede. Tanto a representação das políticas quanto os algoritmos são implementados na linguagem Lisp.

3. Motivação

Vários trabalhos tratam da representação de configuração de *firewalls*. Entretanto, em geral eles tratam de problemas específicos, de forma que poucos deles conseguem abranger todos os aspectos de configuração. Alguns aspectos são a independência de dispositivos, a existência de diferentes tipos de *firewall* na rede, o nível de abstração das linguagens, independência de topologia, entre outros.

O trabalho apresentado em [Guttman 1997] busca representar as regras em uma linguagem de alto-nível independente de topologia, mas é limitado apenas à configuração de *firewalls* mais simples, do tipo filtro de pacotes. O trabalho de DeTreville, representa a configuração em alto nível e também permite criar regras indicando usuários, mas não faz considerações sobre a representação e tratamento das regras de acordo com a topologia. Todos os trabalhos possuem seus pontos fortes, mas há sempre um aspecto que não é explorado.

Em uma rede heterogênea, é bem provável que existam dispositivos com diferentes características e limitações, que não são capazes de aplicar qualquer tipo de

regra. Por exemplo: os *firewalls* mais simples são capazes de identificar apenas a origem e o destino dos pacotes (IP e porta). Outros *firewalls* podem manter o estado das conexões (*stateful firewalls*), decidindo se um pacote é aceito ou não dependendo de pacotes anteriores. *Firewalls* mais novos podem identificar o usuário responsável por um determinado tráfego ou mesmo buscar por informações dentro dos pacotes, identificando protocolos de camadas acima da camada de rede.

Uma regra que diferencie a decisão de acordo com o usuário não poderá ser aplicada em equipamentos mais antigos. Podem existir casos em que a rede possua vários *firewalls*, mas que somente alguns deles possuam essa capacidade. Um exemplo dessa situação pode ser visto na Figura 1.

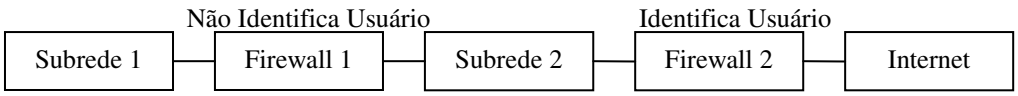

Figura 1: Exemplo de Rede

Nesse exemplo, existe apenas um equipamento (*Firewall* 2) entre as subredes e a internet capaz de identificar o usuário. Então, se for definida uma regra do tipo "O usuário Usr1 pode acessar a internet", e supondo que nenhum outro usuário tenha essa permissão, é necessário configurar o *Firewall* 1 para permitir que todo o trafego da Subrede1 possa ir para a Internet, e configurar o *Firewall* 2 para permitir que apenas o usuário Usr1 tenha acesso a internet a partir da Subrede 1.

Dessa forma, apesar do *Firewall* 1 ser configurado para permitir a passagem de pacotes independente do usuário, o *Firewall* 2 verifica quem é o usuário e controla se este pode ou não acessar a Internet. Aparentemente, a política de segurança não foi violada. Entretanto, em uma política mais complexa, é necessário verificar todas as regras, para certificar que a configuração aplicada ao *Firewall* 1 (permitir todo tráfego da Subrede 1 para a internet) não viola outra regra da política.

Nesse trabalho, será utilizado o termo "capacidade" (*capability*) para indicar um determinado recurso ou funcionalidade do *firewall*. Por exemplo, um *firewall* que seja capaz de identificar o usuário terá a capacidade de identificação de usuário. Outros exemplos de capacidade são: identificação de protocolo de aplicação, manter informações de estado das conexões (*stateful*), identificação de seqüências de caracteres dentro do pacote, entre outros.

4. Proposta

O objetivo deste trabalho é apresentar um *framework* para representação de políticas de segurança para configuração de *firewalls*, composto por uma linguagem para representação de políticas e um mecanismo que interprete e indique as regras que se aplicam a cada equipamento. As principais características são:

- Permitir representar a política de segurança para toda a rede, de forma centralizada, independente da topologia e das tecnologias envolvidas;
- Permitir representar as capacidades de cada *firewall* (por exemplo, um *firewall* possui a capacidade de identificar o usuário, enquanto ouro não a possui);

- Separar e indicar as regras que devem ser aplicadas a cada *firewall*;
- Eventualmente, criar novas regras ou modificá-las de forma que a política possa ser atendida, sem violá-la (nos casos em que algum *firewall* não possua uma capacidade requerida);
- Nos casos em que os *firewalls* não possuam as capacidades necessárias e não for possível alterar a política sem violá-la, indicar que a política não pode ser aplicada, indicando a regra que vai causar a violação quando aplicada.

A Figura 2 apresenta a arquitetura do *framework*. A especificação da política de segurança é feita de forma independente da topologia e dos dispositivos. Com essa separação, torna-se possível modificar a topologia ou os dispositivos existentes na rede sem haver necessidade de modificar a política. Basta processar a política com as novas informações para se obter as regras que devem ser aplicadas aos *firewalls*.

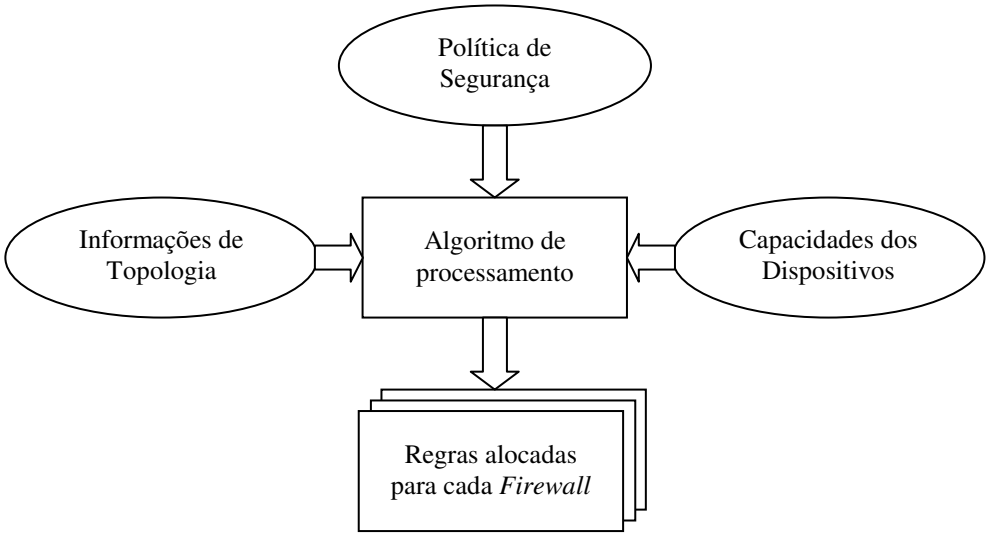

Figura 2: Arquitetura

O Prolog foi escolhido para representar e processar as políticas. Há vários motivos para essa escolha, entre eles o fato de ser uma linguagem lógica, voltada à implementação de algoritmos para realização de inferências e obtenção conclusões a partir dos fatos informados; ser útil para representar e manipular conhecimento; e a existência de interpretadores prontos que podem ser utilizados no processamento da política.

O Prolog é baseado em noções matemáticas de relações e inferência lógica. É uma linguagem declarativa, em que, ao invés de descrever como computar uma solução, um programa consiste de uma base de fatos e regras (relações lógicas) que descrevem as relações existentes na aplicação. A Política de segurança, as informações de topologia, as informações das capacidades dos *firewalls* e as regras alocadas para cada *firewall* apresentadas na Figura 2 são representadas como fatos. O algoritmo para a solução do problema foi implementado como regras do Prolog, que avaliam os fatos existentes e criam novos fatos com o resultado do processamento.

A Figura 3 apresenta a representação da política de segurança, composta pelos usuários, recursos e regras da política de segurança. A representação é a seguinte:

- Os usuários são representados como fatos da linguagem Prolog denominados "user". Internamente, o algoritmo desenvolvido define o usuário denominado "any", que representa qualquer usuário.
- Os recursos são representados por fatos "resource". Além dos servidores, devem ser representadas as subredes existentes na rede e qualquer recurso que seja indicado por alguma das regras.
- As regras da política são representadas por fatos "rule", e podem ser de duas formas: a primeira é indicando o usuário e o recurso ao qual esse usuário tem acesso, independente da origem do usuário (indica que o usuário tem acesso ao recurso de qualquer ponto da rede), e a segunda é indicando o usuário, o ponto origem de acesso e o recurso a ser acessado.

```
user(usuário).

resource(recurso).
resource(subrede).

rule(número, usuário, recurso).
rule(número, usuário, origem, recurso).
```

Figura 3: Representação de Usuários, Recursos e Regras

A Figura 4 apresenta o código com a representação da topologia e das capacidades dos dispositivos. Na representação de topologia são indicados os *firewalls* existentes na rede, o tipo de cada *firewall* e as conexões entre eles e as redes e/ou recursos.

```
firewall(firewall1).
firewall(firewall2).

connected(subnet1, firewall1, custo).
connected(subnet2, firewall2, custo).

type(firewall1, stateless).
type(firewall2, stateful).
```

Figura 4: Representação da Topologia

A representação da topologia é a seguinte:

- O fato "firewall" indica os *firewalls* que estão presentes na rede e que receberão a configuração de acordo com as políticas.
- O fato "connected" indica as conexões entre os *firewalls,* as redes e os recursos, além de indicar o custo da conexão. O custo é usado nos casos em que há mais de um caminho entre a origem e o destino, sendo que o sistema dará prioridade para os caminhos de menor custo.

- O fato “type” indica a capacidade dos *firewalls*. Nesse artigo, são considerados apenas os tipos *stateful* (nesse artigo tratamos especificamente a capacidade de identificar os usuários responsáveis por determinado tráfego) e *stateless* (apenas filtro de pacotes, incapaz de identificar usuários).

A Figura 5 apresenta o resultado do processamento da política, que é representado por 3 fatos:

- O fato “apply_rule” representa as regras que devem ser aplicadas e que foram obtidas diretamente das regras da política de segurança.
- O fato “apply_inferred_rule” representa as regras que foram inferidas, ou seja, regras que foram criadas para poderem ser aplicadas em *firewalls* sem as características necessárias. Tanto esse tipo de regra quando o tipo “apply_rule” indicam a regra da política utilizada, o usuário, a origem, o destino, e o *firewall* ao qual a regra deve ser aplicada.
- O fato “unfeasible_rule” somente aparecerá no resultado do processamento nos casos em que a política não puder ser aplicada adequadamente, ou seja, se ocorrer alguma violação na política devido a limitações dos *firewalls*. Ele indica a regra da política de segurança que causou o problema, além do usuário, origem, destino e *firewall* ao qual a regra deveria ser aplicada.

```
apply_rule(número, usuário, rede, recurso, firewall).

apply_inferred_rule(número, usuário, rede, recurso, firewall).

unfeasible_rule(número, qualquer, rede, recurso, firewall).
```

Figura 5: Resultado do Processamento da Política

Para determinar como as regras devem ser distribuídas pelos *firewalls*, é necessário avaliar a política considerando as capacidades dos *firewalls*, juntamente com a disposição deles na rede. Esse algoritmo foi implementado utilizando o Prolog, sendo implementado na forma de regras desta linguagem. Foram definidas várias regras com a lógica de processamento. As regras foram divididas em 3 grupos, de acordo com os passos abaixo:

1. Determinar os caminhos existentes na rede que são afetados por cada regra, ou seja, por quais equipamentos um determinado pacote pode passar a partir de uma origem até um destino. Por exemplo, uma regra do tipo “O usuário Usr1 pode acessar o Recurso Rsr1” afetará os caminhos existentes entre cada ponto da rede que o usuário Usr1 possa acessar e o Recurso Rsr1.
2. Após a identificação dos caminhos, é necessário percorrer cada um deles, determinando quais regras deverão ser aplicadas a cada equipamento. Caso o equipamento não seja capaz de implementar a regra (por exemplo, não é capaz de identificar um usuário), é necessário determinar uma regra que o *firewall* possa implementar para tentar atender à regra de política.
3. Por fim, depois de determinar as regras a serem aplicadas, elas são reexaminadas para determinar se a política foi atendida corretamente e não foram concedidos direitos além do que a política define. No caso de violação da política, a regra

que não pode ser atendida é apresentada ao administrador, de forma este possa tomar alguma ação para a solução do problema.

A Figura 6 apresenta um trecho do programa em Prolog, com parte da lógica para determinar as regras a serem aplicadas a cada *firewall* (passo 2 acima). Os números à esquerda não pertencem ao programa, foram colocados apenas para facilitar o entendimento do código.

```
 1. create_rules :-
 2.    path(RuleNum, User, Source, Dest, Path, _),
 3.    create_rules(RuleNum, User, Source, Dest, Path, Path).

 4. create_rules(RuleNum, User, Source, Dest, [Node|SeqPath], Path) :-
 5.    \+ firewall(Node),
 6.    create_rules(RuleNum, User, Source, Dest, SeqPath, Path),
 7.    fail.

 8.create_rules(RuleNum, User, Source, Dest, [Node|SeqPath], Path) :-
 9.    firewall(Node),
10.    type(Node, stateful),
11.    \+ clause(apply_rule(_,User, Source, Dest, Node, Path),true),
12.    assert(apply_rule(RuleNum, User, Source, Dest, Node, Path)),
13.    create_rules(RuleNum, User, Source, Dest, SeqPath, Path),
14.    fail.

15.create_rule(RuleNum, User, Source, Dest, [Node|SeqPath], Cam) :-
16.    firewall(Node),
17.    type(Node, stateless),
18.    \+ clause(apply_inferred_rule(_,any, Source, Dest, No, Cam),true),
19.    assert(apply_inferred_rule(RuleNum, any, Source, Dest, No, Cam)),
20.    create_rule(RuleNum, User, Source, Destination, SeqPath, Cam),
21.    fail.
```

Figura 6: Trecho do Programa de Processamento em Prolog

O funcionamento desse trecho de código é o seguinte:

- As linhas de 1 a 3 selecionam cada caminho existente (os caminhos foram determinados em outro trecho do código), e iniciam o processo de determinação das regras que devem ser criadas.
- As linhas de 4 a 7 avaliam os elementos de rede que não são *firewalls* (redes de origem, destino ou intermediárias e os recursos). Nesse caso, nenhuma regra é criada, o elemento é descartado e passa para a avaliação do próximo. O fato "fail" colocado no final indica para o Prolog que ele deve "falhar" quando chegar no último elemento deste caminho e avaliar outros caminhos. O ato de "falhar" não indica problema na avaliação dos elementos; as regras criadas não são perdidas por serem registradas na forma de novos fatos.
- As linhas de 8 a 14 avaliam os *firewalls* do tipo *stateful* (verificado pelas linhas 9 e 10). A linha 11 verifica se já não existe uma regra determinada para o mesmo usuário, origem e destino para esse *firewall*. A linha 12 registra um fato referente à regra que deve ser aplicada a esse *firewall*, e a linha 13 passa para avaliação do próximo elemento do caminho.

- As linhas de 15 a 21 avaliam os *firewalls* do tipo *stateless*. A lógica é semelhante à avaliação dos *firewalls stateful*, exceto que a regra é criada para qualquer usuário.

5. Estudo de caso

O objetivo deste estudo de caso é comprovar a possibilidade de configurar *firewalls* em uma rede a partir de uma política geral de segurança de alto nível, independente de topologia e dispositivos.

A Figura 7 apresenta uma rede que foi utilizada nesse estudo de caso. A rede é composta por 2 subredes, nas quais qualquer usuário pode utilizar os computadores, e 2 recursos aos quais os usuários podem ou não ter acesso a partir dos computadores das duas subredes. Essas redes e esses recursos são interligados através de 2 *firewalls*, que controlarão os acessos permitidos. Um dos *firewalls* é *stateful*, com a capacidade de identificar o usuário responsável pelo tráfego, e o outro é *stateless*, sendo capaz apenas de implementar filtragem de pacotes por endereços IP e portas.

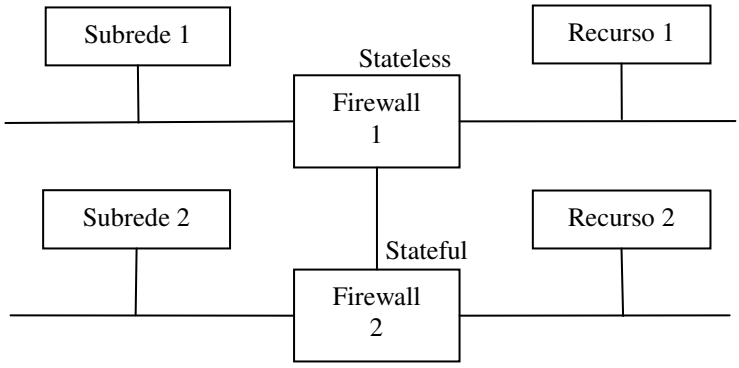

Figura 7: Rede Exemplo

Com base nessa rede, e nos usuários que podem utilizá-la, foi definida uma política de segurança composta pelas seguintes regras:

- O Usuário Usr1 pode acessar o Recurso 1.
- O Usuário Usr1 pode acessar o Recurso 2.
- Qualquer usuário na Subrede 1 pode acessar o Recurso 1.

Essas regras foram representadas na linguagem proposta por três regras, conforme apresentado na Figura 8.

```
rule(1, usr1, recurso1).
rule(2, usr1, recurso2).
rule(3, any, subrede1, recurso1).
```

Figura 8: Representação das Políticas

A Figura 9 apresenta o código com a representação da topologia, junto com a informação dos usuários da rede. A representação da topologia inclui a declaração dos

dois recursos, das duas redes, dos dois *firewalls*, dos tipos de cada um dos *firewalls* e a conexões entre eles. Os comentários são delimitados pelos símbolos "/*" e "*/".

```
/* Representação do Usuário Usr1 */
user(usr1).

/* Representação da Topologia*/
resource(recurso1).
resource(recurso2).
resource(subrede1).
resource(subrede2).

firewall(firewall1).
firewall(firewall2).

connected(subrede1, firewall1).
connected(subrede2, firewall2).
connected(recurso1, firewall1).
connected(recurso2, firewall2).
connected(firewall1, firewall2).

/* Representação das capacidades dos firewalls */
type(firewall1, stateless).
type(firewall2, stateful).
```

Figura 9: Representação da Topologia

O resultado da execução do algoritmo é apresentado na Figura 10.

```
1.  apply_rule(1, usr1, subrede2, recurso1, firewall2).
2.  apply_rule(1, usr1, recurso2, recurso1, firewall2).
3.  apply_rule(2, usr1, subrede1, recurso2, firewall2).
4.  apply_rule(2, usr1, subrede2, recurso2, firewall2).
5.  apply_rule(2, usr1, recurso1, recurso2, firewall2).

6.  apply_special_rule(1, any, subrede1, recurso1, firewall1).
7.  apply_special_rule(1, any, subrede2, recurso1, firewall1).
8.  apply_special_rule(1, any, recurso2, recurso1, firewall1).
9.  apply_special_rule(2, any, subrede1, recurso2, firewall1).
10. apply_special_rule(2, any, recurso1, recurso2, firewall1).
```

Figura 10: Distribuição das Regras pelos *Firewalls*

A interpretação do resultado é o seguinte:

1. A primeira regra do tipo "apply_rule" indica que, processando a regra 1, determinou-se que se deve aplicar ao *Firewall* 2 a permissão para o usuário Usr1 acessar o Recurso 1 a partir da Subrede 2.
2. A primeira regra do tipo "apply_special_rule" indica que, processando a regra 1, determinou-se que se deve aplicar ao *Firewall* 1 a permissão para que qualquer usuário possa acessar o Recurso 1 a partir da Subrede 1.

Nesse exemplo, a política é atendida adequadamente, pois não foi gerada nenhuma informação sobre violação da política. Isso pode ser comprovado da seguinte forma:

- A política diz que todos os usuários da Subrede 1 podem acessar o Recurso 1. Isso é atendido pela regra da linha 6.
- A política permite que apenas o usuário Usr1 possa acessar o Recurso 2 a partir da Subrede 1. Apesar da regra 9 permitir que qualquer usuário possa fazer esse acesso, a regra 3 restringe que somente o usuário Usr1 possa realizar esse acesso. O mesmo ocorre para acessos a partir do Recurso 1 (regras 10 e 5), e para acessos a partir da Subrede 2 (regra 4).
- A política permite que apenas o usuário Usr1 possa acessar o Recurso 1 a partir da Subrede 2. Apesar da regra 7 permitir que qualquer usuário possa fazer esse acesso, a regra 1 restringe que somente o usuário Usr1 possa realizar esse acesso. O mesmo ocorre para acessos a partir do Recurso 2 (regras 8 e 2).

5.1 Avaliação de Desempenho

Para avaliar o impacto causado pelo aumento de elementos (usuários, regras, recursos e *firewalls*), foram realizados diversos testes com diferentes configurações de rede. Para cada um dos testes, foi marcado o tempo necessário para obter-se o conjunto de regras que deve ser aplicado a cada *firewall*.

A primeira avaliação foi realizada da seguinte forma: as regras sempre especificam o usuário, a origem e o destino; o número de *firewalls* é fixo, ou seja, novos recursos são distribuídos pelos *firewalls* existentes, o algoritmo foi executado com 10, 100, 200, 300, 400, 500 e 1.000 regras, e também com 10, 20, 30, 40, 50, 100 e 200 recursos. O Prolog disponibiliza um mecanismo para medição do tempo que é capaz de distinguir o tempo total (entre o início e o fim do processamento) e o tempo efetivo de processamento (o tempo utilizado do processador, desconsiderando outros processos). Todos os gráficos apresentados aqui consideram somente o tempo utilizado para o processamento das regras.

A Figura 11 apresenta os gráficos com os resultados obtidos, sendo que o da esquerda apresenta o tempo gasto no processamento em função do número de recursos, e o da direita apresenta o tempo em função do número de regras. A variação do tempo de processamento com o aumento de recursos apresenta um crescimento não-linear, enquanto que com o aumento do número de regras apresenta um crescimento linear. Pode-se observar que, mesmo para um grande número de recursos e regras (200 recursos e 1.000 regras), foi possível obter o resultado em um tempo relativamente baixo (14,95 segundos).

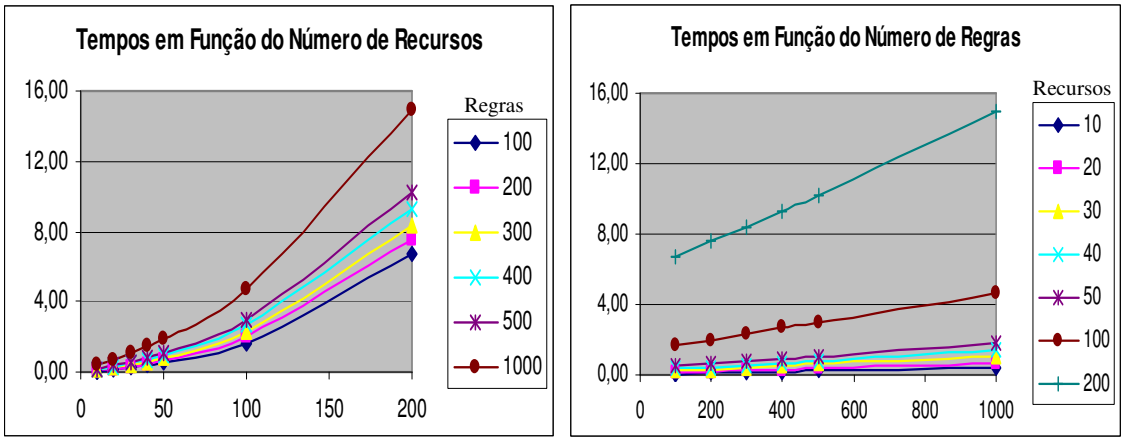

Figura 11: Resultados para Regras Especificando Usuário, Origem e Destino

A segunda avaliação realizada foi baseada nos mesmos parâmetros da primeira, exceto que as regras não especificavam a origem do acesso, ou seja, as regras especificam que o usuário pode acessar determinado recurso a partir de qualquer ponto da rede. Nessa situação, o número de caminhos que devem ser avaliados cresce consideravelmente, tornando o processamento mais lento. A Figura 12 apresenta os gráficos com os resultados obtidos nessa avaliação. Para um cenário grande, com 100 recursos e 500 regras, o tempo de processamento foi de 149,92 segundos.

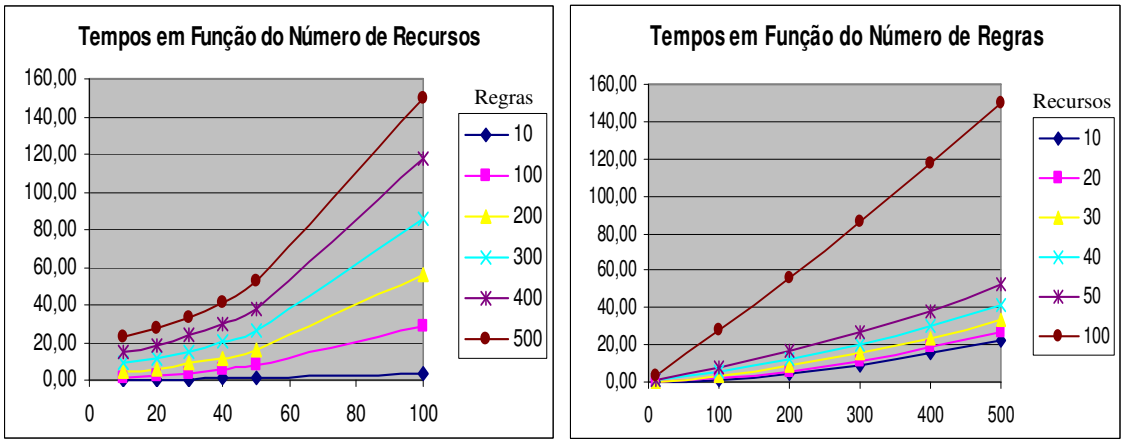

Figura 12: Resultados para Regras Especificando apenas Usuário e Destino

Outra avaliação realizada é com relação ao número de *firewalls* existentes na rede. Para determinar o impacto do aumento desse número, foram consideradas regras indicando o usuário, a origem e o destino, com 10 e 100 recursos e número de regras entre 100 e 1000.

A Figura 13 apresenta os gráficos com os resultados obtidos nessa avaliação. O primeiro gráfico apresenta o tempo de processamento em função do número de *firewalls* existentes na rede para um conjunto de 10 recursos, e o segundo apresenta o tempo para um conjunto de 100 recursos (nos casos em que há mais *firewalls* que recursos, há mais de um *firewall* entre um recurso e outro).

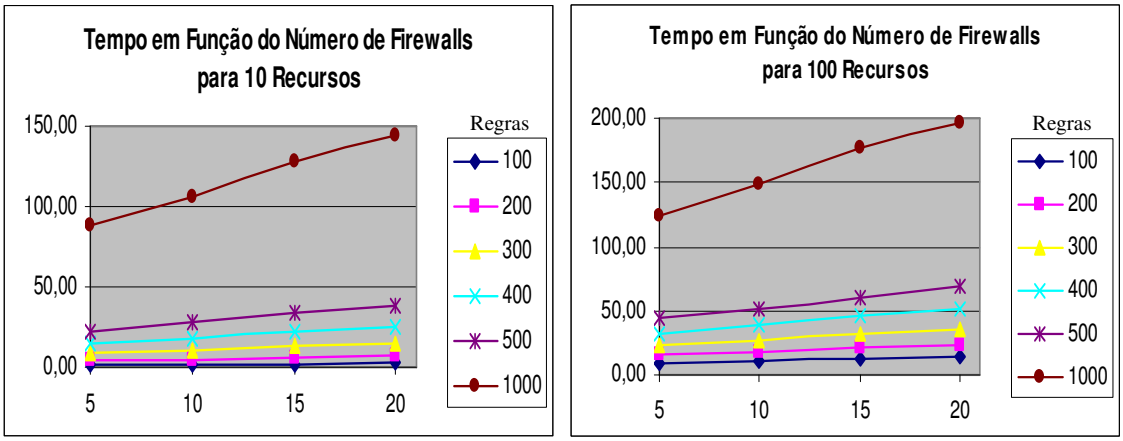

Figura 13: Resultados para um Número Variável de *Firewalls*

Pode-se observar que o aumento do número de *firewalls* não é tão crítico, devido ao crescimento linear no tempo de processamento em função do número de *firewalls*. Para o caso de 1000 regras, 10 recursos e 20 *firewalls*, o tempo de processamento foi de 144,14 segundos, enquanto que para 100 recursos, o tempo foi de 196,34 segundos.

Conclusão

O *framework* apresentado neste artigo é um trabalho em progresso. Os resultados apresentados nesse artigo demonstraram que é possível representar a política de segurança de forma simples, utilizando uma linguagem de alto nível, e que é possível converter essa política em regras de configuração de *firewalls*. Além disso, comprovaram que é possível criar uma representação da política de segurança independente tanto da topologia quanto dos dispositivos existentes na rede.

As avaliações realizadas sobre o *framework* demonstraram que, mesmo para casos com grande número de regras, recursos e *firewalls*, o tempo de processamento não é excessivamente significante para a reconfiguração da rede (o processamento da política com 1000 regras em uma rede com 100 recursos e 20 *firewalls* levou 196,34 segundos).

Novas evoluções estão atualmente em estudo para implementação no *framework*. Entre elas, a inclusão de regras para tratamento de outras capacidades de *firewalls*, visando tornar a solução mais abrangente, e a utilização da Programação Lógica de Restrições (*Constraint Logic Programming*).

Referências

Al-Shaer, E., Hamed, H. (2004) *Discovery of Policy Anomalies in Distributed Firewalls*. 23rd Conference of the IEEE Communications Society (INFOCOMM). Março 2004.

Burns, J., Cheng, A., Gurung, P., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A.V., Martin Jr, D. M. (2001) *Automatic Management of Network Security Policy*. DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume II, pp. 1012. June 2001.

CheckPoint (2005). *Stateful Inspection Technology*. CheckPoint Software Technologies Ltd. Disponível eletronicamente no endereço http://www.checkpoint.com/products/downloads/Stateful_Inspection.pdf

Cisco (2004a). *Cisco PIX Firewall Command Reference*. Cisco Systems Inc. Disponível eletronicamente no endereço http://www.cisco.com/application/pdf/en/us/guest/products/ps3918/c2001/ccmigration_09186a00801cd79b.pdf

Cisco (2004b). *Cisco IOS Reference Guide*. Cisco Systems Inc. Disponível eletronicamente no endereço http://www.cisco.com/warp/public/620/1.pdf

Damianou, N., Dulay, N., et al. (2001). *The Ponder Policy Specification Language*. Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UL, Springer-Verlag.

DeTreville, J. (2002) *Binder, a Logic-Based Security Language*. IEEE Symposium on Security and Privacy. May 2002. pp. 105.

Guttman, J.D. (1997) *Filtering postures: local enforcement for global policies*. 1997 IEEE Symposium on Security and Privacy. May 1997. pp. 0120.

Haixin, D., Jianping, W., Xing, L. (2000) *Policy-Based Access Control Framework for Large Networks*. Eighth IEEE International Conference on Networks (ICON'00). September 2000. pp. 267.

Lee, T.K., Yusuf, S., Luk, W., Sloman, M., Lupu, E., Dulay, N. (2003) *Compiling Policy Descriptions into Reconfigurable Firewall Processors*. 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. April 2003. pp. 39.

Markham, T., Payne, C. (2001) *Security at the Network Edge: A Distributed Firewall Architecture*. DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume I-Volume 1. June 2001. pp. 0279.

Ou, X., Govindavajhala, S., Appel, A. (2004) *Network Security Management with High-level Security Policies*. September 2004.

ROPE (2005). *ROPE - IpTables Scripting Language* – Disponível eletronicamente no endereço http://www.lowth.com/rope/Basics, consultado em Novembro de 2005.