DETOX: Detecção de Inconsistências na Política de Segurança Implementada em Firewall Real

Ygor Kiefer Follador de Jesus Magnos Martinello Eduardo Zambon

[1]NERDS - Núcleo de Estudos em Redes Definidas por Software
UFES - Universidade Federal do Espírito Santo, Campus Goiabeiras

ygor.jesus@ufes.br, magnos@inf.ufes.br, zambon@inf.ufes.br

Abstract. *It is a complex and demanding task to ensure the consistency of rules that implement a network security policy using a firewall. In particular, when such task is not performed properly, network vulnerabilities can arise. In this paper, we present the DETOX tool, a software that detects inconsistencies among rules that constitute a firewall. First, we validate the tool implementation by reproducing and extending the results presented in the literature. After such validation, we perform a real life case study, by analysing the firewall configuration currently in use at a university. During this investigation, the tool discovered several inconsistencies previously unknown.*

Resumo. *Garantir a consistência das regras que implementam uma política de segurança de rede através de um* firewall *é uma tarefa complexa e exaustiva, podendo gerar vulnerabilidades na rede quando mal executada. Neste artigo, realizamos um estudo de teorias e algoritmos já existentes nesta área e apresentamos o DETOX, uma ferramenta para a detecção de inconsistências entre regras que compõem um* firewall*. Além de validar sua implementação reproduzindo os resultados obtidos na literatura, a ferramenta foi utilizada em um caso de estudo real, aplicado sobre a configuração de* firewall *em uso em uma instituição de ensino superior. Neste caso de estudo, a ferramenta indicou a existência de inconsistências que não haviam sido previamente descobertas.*

1. Introdução

Com o aumento do número de pessoas com acesso à Internet, a quantidade de informações trafegando pela rede vem crescendo de maneira exponencial. Com isso, aumenta também a quantidade de ameaças e ataques às redes de computadores, não só às de grandes corporações como também às de pequenos negócios ou domésticas. Consequentemente, a habilidade para controlar tais redes especificando e aplicando políticas de segurança tem ganhado interesse na comunidade de pesquisa em redes, a qual vem buscando maneiras de garantir que essas políticas sejam corretamente aplicadas.

Essa habilidade vem sendo aprimorada com o surgimento de diferentes arquiteturas que apoiam a implantação de políticas de segurança. Tais políticas, em geral, são expressadas através de normas definidas por um analista de redes, que indicam as ações a serem tomadas para cada fluxo de dados presente na rede. Assim, há a necessidade de se operacionalizar essas normas e o mecanismo mais comum para fazê-lo atualmente é conhecido como *firewall*.

Um *firewall* protege uma rede de comunicação determinando os fluxos de dados que podem transitar através dele. Essa filtragem é realizada com base nas diferentes características do fluxos de dados presentes, como, por exemplo, a origem ou o destino dos dados sendo transmitidos. Para isso, um *firewall* é composto de regras que especificam quais dessas características são consideradas seguras ou não, refletindo as normas definidas pelo analista de redes.

Frequentemente, tais regras são especificadas por diferentes analistas em diferentes momentos e isso aumenta a chance do surgimento de conflitos entre elas. Por exemplo, uma nova regra, introduzida por um novo analista, pode acidentalmente permitir tráfego malicioso previamente bloqueado por outro analista. Tais conflitos causam *inconsistências* no comportamento do *firewall* e esse é um aspecto que vem, gradualmente, sendo alvo de novas pesquisas científicas.

Na literatura, a grande maioria de trabalhos atuais relacionados aos *firewalls* tem seu foco na otimização. Outro foco de pesquisa está na busca e na análise da consistência das regras que compõem o *firewall*, entretanto os trabalhos existentes não disponibilizam as implementações das propostas, portanto seus resultados não são reproduzíveis.

Esse artigo propõe o DETOX, uma implementação *open source*[1] de análise e busca de inconsistências na política de segurança implementada e validada em um *firewall* existente. As definições formais de inconsistências entre regras e o algoritmo aqui proposto foram baseados no artigo de [Al-Shaer and Hamed 2004]. Destaca-se como contribuições principais deste trabalho:

- Implementação e disponibilização do DETOX com seu código-fonte a fim de permitir a reproducibilidade dos testes e sua aplicação na prática.
- Correção de erros no algoritmo original de [Al-Shaer and Hamed 2004], além de sua extensão e adaptação para o DETOX.
- Aplicação do método de detecção de inconsistências em um *firewall* real.

Com relação ao terceiro ponto de contribuição, o DETOX foi aplicado ao *firewall* da rede acadêmica corporativa da Universidade Federal do Espírito Santo (UFES). Neste estudo foram descobertas inconsistências na configuração do *firewall* que eram previamente desconhecidas. Estas inconsistências foram apresentadas aos analistas de redes responsáveis para reparação, o que indica a aplicabilidade da ferramenta na prática em casos reais. Além disso, outro aspecto interessante a se destacar é o desempenho da ferramenta: a detecção das regras inconsistentes no caso de estudo (de tamanho considerável) leva apenas alguns milissegundos.

O restante do artigo está organizado como a seguir. A seção 2 fornece uma breve fundamentação teórica sobre inconsistências entre regras e a seção 3 indica trabalhos relacionados e justifica a necessidade da criação do DETOX. A seguir, a seção 3.1 apresenta uma análise aprofundada do algoritmo proposto por [Al-Shaer and Hamed 2004], indicando, em particular, discrepâncias encontradas no material original. A seção 4 descreve a implementação do DETOX, incluindo as correções para as discrepâncias citadas na seção anterior. A validação dessa implementação é discutida na seção 5.1. A aplicação do DETOX no caso de estudo real, o *firewall* da UFES, é apresentada na seção 5.2. Por fim, a seção 6 discorre sobre as conclusões e trabalhos futuros.

[1]Disponível em `https://github.com/llKieferll/DETOX`

2. Fundamentação Teórica

Um *firewall* é composto por regras que analisam e decidem se determinados fluxos de dados são considerados seguros ou não. Por sua vez, uma regra é composta por campos, onde cada campo refere-se a uma característica existente em cada fluxo de dados presente no *firewall* como, por exemplo, o IP de destino ou a porta de origem. Essa análise é realizada comparando cada campo de uma regra (o IP de origem definido na regra) à característica correspondente do fluxo sendo analisado (o IP de origem do fluxo). Caso ocorra um *match* entre todos os campos de uma regra referentes às características de um fluxo e o fluxo em si, é aplicada a ele a ação estabelecida por tal regra, normalmente presente em um campo denominado "Action". A existência desse campo é obrigatória e ele define se o fluxo definido pela regra é aceito ou bloqueado. Considere como exemplo uma regra fictícia a seguir, formada por três campos, IP de origem, IP de destino e *Action*:

$$10.0.0.1 \| * . * . * . * \| deny$$

Essa regra define que qualquer tráfego cuja origem é um IP específico (10.0.0.1) e cujo destino seja qualquer IP (*.*.*.*), deve ser negado (*deny*). Convém destacar que as regras são analisadas de forma sequencial, ou seja, uma regra anterior tem prioridade maior que uma regra posterior e apenas o primeiro *match* entre regra e fluxo é aplicado. Portanto, a inserção de uma nova regra como abaixo gera uma inconsistência pois a regra nova terá prioridade maior que a regra antiga e, como os campos referentes às características de fluxo são iguais, a regra antiga nunca será aplicada. Logo, o tráfego que antes era considerado malicioso agora será sempre aceito.

$$10.0.0.1 \| * . * . * . * \| accept$$
$$10.0.0.1 \| * . * . * . * \| deny$$

As subseções seguintes apresentam as definições das regras que compõem o *firewall* de exemplo a ser usado por este trabalho, bem como as definições das possíveis relações entre regras e as inconsistências que tais relações podem causar.

2.1. Relações entre regras

Nesta seção, apresentamos as definições das relações existentes entre duas regras. O *firewall* de exemplo de [Al-Shaer and Hamed 2004] foi também utilizado para facilitar a visualização de tais relações, desde o início da análise do algoritmo proposto, até a validação do DETOX. As regras que o compõem e seus campos encontram-se na Figura 1.

Considere que uma regra R possui n campos, para qualquer valor positivo de n. Utilizamos $R[i], 0 \leq i < n$ para indicar o campo i da regra R.

Definição 1: Duas regras R_x e R_y são **completamente disjuntas** (relação denotada como $R_x \Delta_{CD} R_y$) se:

$$\forall i : R_x[i] \not\bowtie R_y[i] \text{ onde } \bowtie \in \{\subset, \supset, =\}, i \neq \text{Action}.$$

Ou seja, duas regras são completamente disjuntas quando não há nenhuma relação de igualdade ou de interseção entre *todos* os campos de ambas regras.

	Protocol	Source Address	Source Port	Destination Address	Destination Port	Action
1:	tcp,	140.192.37.20,	any,	*.*.*.*,	80,	deny
2:	tcp,	140.192.37.*,	any,	*.*.*.*,	80,	accept
3:	tcp,	*.*.*.*,	any,	161.120.33.40,	80,	accept
4:	tcp,	140.192.37.*,	any,	161.120.33.40,	80,	deny
5:	tcp,	140.192.37.30,	any,	*.*.*.*,	21,	deny
6:	tcp,	140.192.37.*,	any,	*.*.*.*,	21,	accept
7:	tcp,	140.192.37.*,	any,	161.120.33.40,	21,	accept
8:	tcp,	*.*.*.*,	any,	*.*.*.*,	any,	deny
9:	udp,	140.192.37.*,	any,	161.120.33.40,	53,	accept
10:	udp,	*.*.*.*,	any,	161.120.33.40,	53,	accept
11:	udp,	140.192.38.*,	any,	161.120.35.*,	any,	accept
12:	udp,	*.*.*.*,	any,	*.*.*.*,	any,	deny

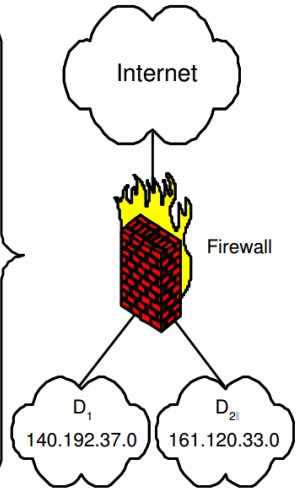

Figura 1. *Firewall* de exemplo de acordo com [Al-Shaer and Hamed 2004]

Definição 2: Duas regras R_x e R_y são **parcialmente disjuntas** ($R_x \Delta_{PD} R_y$) se:

$$\exists i, j \text{ tal que } R_x[i] \bowtie R_y[i] \text{ e } R_x[j] \not\bowtie R_y[j]$$
$$\text{onde } \bowtie \in \{\subset, \supset, =\}, i, j \neq \text{Action}, i \neq j.$$

Como exemplo de regras parcialmente disjuntas temos as regras 2 e 6 na Figura 1: todos os campos de ambas as regras coincidem, exceto o campo da porta de destino.

Definição 3: Duas regras R_x e R_y são um ***matching* exato** ($R_x \Delta_{EM} R_y$) se:

$$\forall i : R_x[i] = R_y[i] \text{ onde } i \neq \text{Action}.$$

Ou seja, duas regras possuem um *matching* exato quando todos os campos exceto o campo *Action* forem iguais.

Definição 4: Duas regras R_x e R_y são um ***matching* inclusivo** ($R_x \Delta_{IM} R_y$) se:

$$\forall i : R_x[i] \subseteq R_y[i] \text{ e } \exists j \text{ tal que } R_x[j] \neq R_y[j] \text{ onde } i, j \neq \text{Action}.$$

Como exemplo de regras com *matching* inclusivo temos as regras 1 e 2 na Figura 1.Vale destacar que o quantificador existencial na definição acima é necessário para garantir que o *matching* não é exato.

Definição 5: Duas regras R_x e R_y são **correlacionadas** ($R_x \Delta_C R_y$) se:

$$\forall i : R_x[i] \bowtie R_y[i] \text{ e } \exists j, k \text{ tal que } R_x[j] \subset R_y[j] \text{ e } R_x[k] \supset R_y[k]$$
$$\text{onde } \bowtie \in \{\subset, \supset, =\}, i, j, k \neq \text{Action}, j \neq k.$$

Como exemplo de regras correlacionadas temos as regras 1 e 3 na Figura 1, pois os campos de endereço de origem e destino estão contidos um no outro nessas regras.

2.2. Inconsistências

Baseado nas relações apresentadas na seção anterior, definimos nesta seção as possíveis *inconsistências* causadas por essas relações. Tais definições também são apresentadas

no trabalho de [Al-Shaer and Hamed 2004]. Considere que, para uma regra R qualquer, utilizamos $R[Action]$, para indicar o campo referente à ação a ser tomada pela regra R (aceitar ou bloquear) em relação ao fluxo também definido pela regra R. Assim, temos:

Shadowing: Uma regra R_y é sombreada por uma regra de prioridade maior R_x se uma das seguintes condições é satisfeita:

$$R_x \Delta_{EM} R_y, R_x[Action] \neq R_y[Action] \quad (1)$$
$$R_y \Delta_{IM} R_x, R_x[Action] \neq R_y[Action] \quad (2)$$

Exemplo: regras 4 e 3 na Figura 1. A regra 4 possui uma relação de *matching* inclusivo com a regra 3, que possui prioridade maior.

Correlation: Duas regras R_x e R_y causam uma inconsistência de correlação se a seguinte condição é satisfeita:

$$R_x \Delta_C R_y, R_x[Action] \neq R_y[Action] \quad (3)$$

Exemplo: regras 1 e 3 na Figura 1.

Generalization: Uma regra R_y é uma generalização de uma regra de prioridade maior R_x se a seguinte condição é satisfeita:

$$R_x \Delta_{IM} R_y, R_x[Action] \neq R_y[Action] \quad (4)$$

Exemplo: regras 2 e 1 na Figura 1, onde a regra 2 generaliza a regra 1.

Redundancy: Uma regra R_y é redundante à uma regra de prioridade maior R_x se uma das seguintes condições é satisfeita:

$$R_x \Delta_{EM} R_y, R_x[Action] = R_y[Action] \quad (5)$$
$$R_y \Delta_{IM} R_x, R_x[Action] = R_y[Action] \quad (6)$$

Exemplo: regras 7 e 6 na Figura 1, onde a regra 7 é redundante em relação à regra 6.

Além disso, uma regra R_x é redundante à uma regra de prioridade menor R_y se a seguinte condição é satisfeita:

$$\begin{gathered} R_x \Delta_{IM} R_y, R_x[Action] = R_y[Action] \\ \text{e } \nexists R_z \text{ tal que } prioridade(R_x) > prioridade(R_z) > prioridade(R_y), \\ R_x\{\Delta_{IM}, \Delta_C\} R_z, R_x[Action] \neq R_z[Action] \end{gathered} \quad (7)$$

Exemplo: regras 4 e 8 na Figura 1. Convém destacar que esta última condição (7) é apresentada, mas não implementada no artigo original de [Al-Shaer and Hamed 2004].

3. Trabalhos Relacionados

Nos trabalhos que têm seu foco na análise e deteção de inconsistências, [Eppstein and Muthukrishnan 2001] e [Hari et al. 2000] fazem uma análise limitando-se apenas à um tipo de inconsistência existente, a de *correlação*. Já os trabalhos de [Bartal et al. 1999] e [Mayer et al. 2000] propõem uma política em linguagem de alto nível para definir, analisar e em seguida mapear novas regras de filtragem, mas como

a maior parte dos *firewalls* utilizados hoje em dia contém regras de baixo nível, redefinir políticas já existentes em linguagens de alto nível torna-se um esforço muito grande frente ao benefício conquistado. Além disso, as ferramentas para análise propostas nestes artigos ("*Firmato*" e "*Fang*", respectivamente) não parecem estar mais disponíveis.

Já [Khorchani et al. 2012] propõem o uso de *model checking* para a análise e detecção de inconsistências usando *Visibility Logic (VL)*, um tipo de lógica multi-modal. Os autores argumentam que as definições de inconsistências apresentadas na seção 2.2 podem ser vistas como casos particulares de fórmulas em VL, e descrevem a implementação de um *model checker* dedicado para VL. Infelizmente, esta implementação também não está disponível. Mais recentemente, [Mukkapati and Ch.V.Bhargavi 2013] realizam um trabalho interessante utilizando álgebra relacional para a detecção de inconsistências, definindo o que são e como podem ser encontradas, juntamente com a ideia de combinação de regras similares a fim de diminuir seu número total e otimizar o *firewall*. Entretanto, o trabalho também não apresenta um algoritmo ou uma ferramenta.

Merece destaque o trabalho de [Yuan et al. 2006], que fornece uma pesquisa mais completa, apresentando regras cujas ações aplicáveis sobre os fluxos de dados que vão além de *accept* e *deny* e analisando a relação existente entre três ou mais regras, fornecendo assim uma perspectiva de trabalhos futuros para o DETOX. Entretanto, novamente, o *toolkit* "Fireman" proposto não foi encontrado, nem qualquer código relacionado a ele, o quê inviabiliza o seu uso. Esta é a mesma situação da ferramenta original criada por [Al-Shaer and Hamed 2004], chamada de *Firewall Policy Advisor* (FPA).

Um vez que nenhuma das ferramentas similares propostas na literatura se encontram mais disponíveis, buscou-se o desenvolvimento do DETOX para preencher esta lacuna. Convém destacar que a indisponibilidade das demais ferramentas impede um estudo comparativo entre o DETOX e as demais.

3.1. Discrepâncias do algoritmo original de [Al-Shaer and Hamed 2004]

A base do trabalho aqui descrito iniciou-se com a análise do algoritmo em pseudo-código apresentado por [Al-Shaer and Hamed 2004]. Imediatamente percebe-se quatro discrepâncias entre o algoritmo sugerido e os resultados apresentados no artigo [Al-Shaer and Hamed 2004]. A condição (7) da seção anterior não foi implementada no algoritmo sugerido. Na verdade, caso a primeira linha da condição (7) seja satisfeita, o caso é imediatamente considerado como *Shadowing* sem análise das condições seguintes. Além disso, a condição (2) é considerada como *Generalization* quando trata-se de um caso de *Shadowing* e a condição (4) é considerada como *Shadowing* quando trata-se de um caso de *Generalization*.

Apesar disso, os resultados finais em [Al-Shaer and Hamed 2004] aparentavam estar corretos e não condiziam com o algoritmo sugerido. Decidiu-se então realizar uma análise manual do *firewall* com base nas definições da seção anterior a fim de validar os resultados finais. Por fim, todos eles estavam corretos, exceto por uma redundância entre as regras 4 e 8, a qual não foi detectada e também uma generalização entre as regras 12 e 11, embora essa última possa não ter sido documentada devido à regra 11 ser um caso de inconsistência de irrelevância. Entretanto, os resultados apresentados pelo algoritmo sugerido estavam, em sua grande maioria, incorretos. Devido às discrepâncias encontradas, decidiu-se implementar o DETOX para corrigi-las, como descrito a seguir.

4. Desenvolvimento

O funcionamento e uso do DETOX pode ser dividido em quatro etapas. A primeira etapa refere-se ao preenchimento de um arquivo de configuração que reflete as configurações do *firewall* a ser analisado, necessários para o funcionamento do DETOX e explicado na subseção 4.1. A subseção 4.2 lida com a leitura das regras, sendo esta realizada de acordo com os dados fornecidos pelo arquivo de configuração devidamente preenchido. Em seguida, o tratamento necessário é aplicado às regras compostas encontradas transformando-as em regras simples, o que será explicado na seção 4.3. Por fim, na seção 4.4 apresentamos o algoritmo de busca de inconsistências utilizado pelo DETOX. Maiores detalhes sobre os itens citados em cada subseção seguinte podem ser encontrados no site[2] da ferramenta.

4.1. Arquivo de Configuração

O arquivo de configuração deve ser preenchido pelo usuário e conter informações básicas para o funcionamento do DETOX. As informações mais básicas incluem o nome de cada campo existente em uma regra e quais destes campos serão incluídos na busca por inconsistências. Além disso, diferentes campos geralmente são comparados de diferentes maneiras. Tome como exemplo endereços de rede e interfaces. Endereços podem ser diferentes, iguais ou ainda estarem contidos um no outro. Já interfaces podem apenas ser diferentes ou iguais.

Por isso, é necessário que seja fornecido também um método para comparar cada par de campos correspondentes entre duas regras para que seja possível descobrir a relação entre eles (tais métodos não são necessáriamente distintos). Para fornecer maior facilidade de uso, quatro métodos iniciais que abrangem a maior parte dos campos mais utilizados em uma regra já constam como métodos *default* no arquivo. Com a ferramenta sendo implementada em Java, todos esses dados foram organizados em estruturas do tipo *enums*, a fim de promover a modularização dos dados e a flexibilidade necessária para adaptar o arquivo a qualquer tipo de *firewall* ou qualquer outro ambiente que utilize regras em formato similar.

4.2. Leitura das regras

Com o arquivo de configuração definido, a leitura das regras do *firewall* em questão pode ser realizada. Ressaltamos que tais regras podem conter um campo, não obrigatório, cuja função é unicamente definir se encontram-se habilitadas ou desabilitadas no *firewall*; esse campo é chamado de "*Enabled*". As regras também devem conter, obrigatoriamente, o campo "*Action*", apresentado na seção 2. Portanto, deve-se verificar previamente a existência de ambos a fim de garantir que a estrutura definida no arquivo de configuração está correta, bem como garantir que somente regras pertinentes serão lidas.

Após essa verificação, seguida da leitura, tem-se uma lista contendo as regras ativas, as quais, nessa lista, são compostas apenas pelos campos que devem ser analisados na busca por inconsistências, como definido no arquivo de configuração explicado na seção anterior. A estrutura de lista permite manter a ordem (prioridade) das regras fornecendo um acesso imediato a elas. No entanto, os campos podem conter dois ou mais valores distintos e agrupados, geralmente separados por vírgula. Tais campos são chamados compostos e devem ser transformados em campos simples, como é explicado a seguir.

[2]Disponível em `https://github.com/llKieferll/DETOX`

4.3. Simplificação de regras

Os campos de uma regra podem possuir um único elemento representando um valor, um único elemento representando uma faixa de valores ou ainda dois ou mais elementos distintos, podendo representarem tanto valores únicos como também faixas de valores. Uma regra onde todos os seus campos contém um único elemento (mesmo que tal elemento seja uma faixa de valores) é denominada *regra simples*. Tome como exemplo, regras formadas por três campos, IP de origem, IP de destino e porta, nessa ordem. As seguintes regras são exemplos de regras simples:

1. 10.0.0.1 || 10.0.1.1 || 50
2. 10.0.0.1 || 10.0.1.1 || 51
3. 10.0.0.1 || 10.0.1.1 || 50-55
4. 10.0.0.* || 10.0.1.* || *

Note que, nos exemplos citados acima, o campo referente à porta na regra 3, *"50-55"*, é considerado simples pois é um elemento único que representa uma faixa de valores. Todos os campos das regras citadas possuem um único elemento tornando-as regras simples. No caso de uma regra conter dois ou mais elementos, teremos diferentes regras condensadas em apenas uma. Esse tipo de regra é denominada *regra composta*. Seguindo o mesmo padrão anterior, são exemplos de regras compostas:

5. 10.0.0.1 || 10.0.1.1 || 50, 51
6. 10.0.0.* || 10.0.1.*, 10.0.2.* || *
7. 10.0.0.1, 10.0.0.2 || 10.0.1.1, 10.0.1.2|| 50

Note que o campo referente à porta na regra 5 possui dois valores distintos (50 e 51) e essa regra deve ser simplificada. Ao fazê-lo duas novas regras são geradas, uma para cada elemento distinto, para cada campo composto. Os campos simples são repetidos e os elementos distintos são distribuídos em suas respectivas posições nas regras geradas, consequentemente levando às regras simples 1 e 2. De maneira similar, a simplificação da regra 6 leva às seguintes regras simples:

- 10.0.0.* || 10.0.1.* || *
- 10.0.0.* || 10.0.2.* || *

E a simplificação da regra 7 leva às seguintes regras simples:

- 10.0.0.1 || 10.0.1.1 || 50
- 10.0.0.1 || 10.0.1.2 || 50
- 10.0.0.2 || 10.0.1.1 || 50
- 10.0.0.2 || 10.0.1.2 || 50

Com base nesse conhecimento, é realizada a simplificação das regras compostas. Analisamos regra por regra, em ordem, verificando os campos de cada uma e, caso sejam todos simples, a armazenamos em uma estrutura, como citado na seção anterior, destinada a conter apenas regras simples. Caso uma regra composta seja encontrada, devemos simplificá-la. Todas as regras simples geradas são transferidas juntas para a lista de regras a fim de ocuparem conjuntamente a posição da regra composta que as gerou. Assim, todas as regras geradas têm a mesma prioridade da regra que as gerou.

4.4. Busca por inconsistências

Com a certeza que todas as regras compostas foram transformadas em regras simples, inicia-se então a busca por inconsistências entre elas. O algoritmo 1 representa a versão final que foi implementada no DETOX, incluindo as adaptações que corrigem as discrepâncias citadas na seção 3.

Algorithm 1 Inconsistency Finder

```
1:  procedure FIND
2:      Input: rules - ArrayList
3:      Output: incons_list - HashMap

4:      active_fields ← Lista de campos ativos definidos no arquivo de configuração
5:      incons_list ← {}
6:      Remover "Enabled" de active_fields
7:      Remover "Action" de active_fields
8:      for Cada R_x em rules do
9:          for Cada R_y em rules onde prioridade(R_x) > prioridade(R_y) do
10:             rules_relation ← NONE
11:             for Cada field em active_fields do
12:                 fields_relation ← field.Compare(R_x[field], R_y[field])
13:                 rules_relation ← Rel(rules_relation, fields_relation)
14:             if rules_relation ≠ DISJOINT then
15:                 switch rules_relation
16:                     case CORRELATED do
17:                         if R_x[Action] ≠ R_y[Action] then
18:                             Adicionar (R_x, CORRELATION, R_y) em incons_list
19:                     case SUPERSET do
20:                         if R_x[Action] = R_y[Action] then
21:                             Adicionar (R_y, REDUNDANCY, R_x) em incons_list
22:                         else
23:                             Adicionar (R_y, SHADOWING, R_x) em incons_list
24:                     case EXACT do
25:                         if R_x[Action] = R_y[Action] then
26:                             Adicionar (R_y, REDUNDANCY, R_x) em incons_list
27:                         else
28:                             Adicionar (R_y, SHADOWING, R_x) em incons_list
29:                     case SUBSET do
30:                         if R_x[Action] = R_y[Action] then
31:                             XZ_fields_rel ← NONE
32:                             XZ_rules_rel ← NONE
33:                             Z_exists ← false
34:                             for Cada regra R_z entre R_x e R_y do
35:                                 for Cada field em active_fields do
36:                                     XZ_fields_rel ← field.Compare(R_x[field],R_z[field])
37:                                     XZ_rules_rel ← Rel(XZ_rules_rel, XZ_fields_rel)
38:                                 if rules_relation ∈ CORRELATED,SUBSET then
39:                                     if R_x[Action] ≠ R_z[Action] then
40:                                         Z_exists ← true
41:                             if Not Z_exists then
42:                                 Adicionar (R_x, REDUNDANCY, R_y) em incons_list
43:                         else
44:                             Adicionar (R_y, GENERALIZATION, R_x) em incons_list
```

Inicialmente, removemos os campos *Enabled*, pois não tem influência na relação entre regras, e *Action*, pois influencia apenas o tipo de inconsistência, caso exista, decorrente da relação encontrada entre as regras. Devemos então analisar os pares de regras R_x e R_y, onde a prioridade de R_y é sempre menor que a prioridade de R_x.

A relação momentânea inicial entre duas regras é inexistente. A cada comparação de campos correspondentes de duas regras sendo analisadas (linha 12 do Algoritmo 1), atualizamos a relação momentânea entre as regras em questão, com base na combinação entre a relação momentânea existente e o resultado da comparação entre os campos analisados (linha 13 do Algoritmo 1) através do Algoritmo 2. Cada nova comparação de campos subsequentes irá atualizar a relação momentânea até que todos os campos das regras sendo analisadas sejam comparados e a relação final entre tais regras seja encontrada.

No Algoritmo 2, realizamos a atualização da relação momentânea entre duas regras com base na ultima comparação entre campos realizada e a relação momentânea atual entre elas. Para tal, verificamos qual a relação encontrada entre entre tais campos (linha 4 do Algoritmo 2). Em seguida, verificamos a relação momentânea atual entre as regras (linhas 6, 8, 11, 13, 16 e 18 do Algoritmo 2). A combinação dessas duas informações nos fornece uma nova relação momentânea entre as regras sendo analisadas.

Algorithm 2 Rules Relation Finder

```
 1: procedure REL
 2:     Input: rules_relation - Int, field_relation - Int
 3:     Output: <Relation between rules> - Int

 4:     switch field_relation
 5:         case EQUALS do
 6:             if rules_relation = NONE then
 7:                 return EXACT
 8:             else
 9:                 return rules_relation
10:         case CONTAINS do
11:             if rules_relation = {SUBSET,CORRELATED} then
12:                 return CORRELATED
13:             else if rules_relation ≠ DISJOINT then
14:                 return SUPERSET
15:         case CONTAINED do
16:             if rules_relation = {SUPERSET,CORRELATED} then
17:                 return CORRELATED
18:             else if rules_relation ≠ DISJOINT then
19:                 return SUBSET
20:         case default do
21:             return DISJOINT
```

Após todos os campos do par de regras em questão serem analisados, encontraremos a relação final entre as regras. Caso sejam disjuntas (linha 14) não há inconsistências. Caso contrário, devemos analisar sua relação final (linha 15 do Algoritmo 1) e os campos *Action* de ambas as regras (linhas 17, 20, 25, 30 e 39 do Algoritmo 1). Com isso, encontramos o tipo de inconsistência existente entre elas, a qual deve ser armazenada na lista de inconsistências. Terminado o procedimento para todos os pares de regras possíveis, temos como resultado uma *HashMap* que contém a todas as inconsistências encontradas.

5. Testes realizados

Com o novo algoritmo implementado, realizamos a sua validação e, após esta, sua aplicação em um caso de estudo real. Para a validação, utilizamos o *firewall* de exemplo de [Al-Shaer and Hamed 2004], exibido na Figura 1, a fim de replicar os resultados esperados de acordo com teste manual realizado na seção 3. Como caso de estudo real, utilizamos a rede acadêmica da UFES, como descrito na seção 1. As subseções seguintes tratam da validação e da aplicação em caso de estudo real.

5.1. Validação

Para a validação do DETOX, realizamos uma comparação entre o resultado fornecido pela versão final do nosso algoritmo, com os resultados encontrados durante nosso teste manual, como realizado na seção 3. Os resultados apresentados pelo DETOX foram exatamente os resultados esperados, sendo iguais aos resultados expostos pelo teste manual e realizando duas detecções adicionais em relação aos resultados obtidos por [Al-Shaer and Hamed 2004], sendo uma *redundancy* e uma *generalization* entre as regras 4 e 8 e as regras 12 e 11, respectivamente, da Figura 1. Os resultados pertinentes são apresentados como na Tabela 3, omitindo as regras que não apresentam inconsistências.

Tabela 1. Inconsistências no *firewall* exemplo

Rule No	CORRELATED	GENERALIZATION	REDUNDANT	SHADOWED
1	{3}			
2		{1}		
4			{8}	{2, 3}
5	{7}			
6		{5}		
7			{6}	
8		{2, 3, 6, 7}		
9			{10}	
12		{9, 10, 11}		

5.2. Caso de estudo

Após realizada a validação do DETOX, buscamos aplicá-lo em um caso de estudo real, a fim de obter resultados também reais e verificar sua viabilidade. Para isso, utilizamos o *firewall* do Núcleo de Tecnologia de Informação (NTI) da UFES. Verificamos quais campos deveriam ser analisados na busca por inconsistências e quais campos poderiam ser ignorados e alteramos o arquivo de configuração para refletir os campos que compõem o *firewall*. Os métodos *default* de comparação foram suficientes para abranger todos os campos inclusos na busca por inconsistências.

A definição desse *firewall* contém 207 regras, todas contendo o campo *Enabled* e sendo várias delas compostas. Dessas 207 regras, 174 estavam habilitadas. Das habilitadas, após a simplificação conforme descrito na seção 4.3, obtemos um total de 317 regras simples. Porém, muitas delas continham *aliases* (um nome dado a uma rede ou endereço

de IP ao invés do endereço propriamente dito). Por questões de anonimidade, tais *aliases* foram substituídos por IPs e redes fictícios, mas de modo a manter quaisquer relações existentes entre os IPs reais sem comprometer os resultados. Os resultados obtidos através da aplicação do DETOX encontram-se na Tabela 2:

Tabela 2. Inconsistências no *firewall* do NTI

Rule No	CORRELATED	GENERALIZATION	REDUNDANT
12			{46}
13			{21,46}
14			{25}
15			{24}
18	{22,23,42,45,55,57}		
19			{46}
20			{46}
21			{46}
31			{52}
35			{54}
41			{46}
48			{46}
60			{24}
61			{25}
72	{301}		
81			{71}
82			{71}
83			{71}
84			{71}
85			{125}
125	{301}		
187		{186}	
188		{184}	
189		{185}	
289	{301}		
Total de Inconsistências	**9**	**3**	**19**
Regras Inconsistentes	**11 de 317**	**6 de 317**	**25 de 317**
Porcentagem de Inconsistências	**3.47%**	**1.89%**	**7.88%**

Observe que não há uma coluna referente à inconsistência do tipo *shadowing*. Isso ocorre porque não foram encontradas inconsistências desse tipo, portanto, a omitimos nos resultados. Encontramos que, das 317 regras simples, 41 regras distintas apresentaram algum tipo de inconsistência, o que representa cerca de 12,93% do total de regras existentes no *firewall*. É fácil notar que, como citado na seção 1, mesmo um *firewall* administrado por pessoas experientes e competentes pode ser comprometido sem o suporte adequado para garantir a consistência da política de segurança aplicada pelas regras de um *firewall*.

Para exemplificar o caso real, exibimos aqui a composição da regra 18 e suas inconsistências de *correlation*, a regra 81 e sua inconsistência de *redundancy* e a regra 187 e sua inconsistência de *generalization*. Os campos *hits*, *logging*, *description* e *time* não foram incluídos na análise pois referenciavam características meramente informativas.

Tabela 3. Exemplo de regras inconsistentes no *firewall* real

ID Pos-Simp	Interface	Source	Destination	Service	Action	ID Pre-Simp
18	inside	any	10.10.2.*	ip	deny	10
22	inside	10.20.3.*	any	ip	permit	13
23	inside	10.20.4.*	any	ip	permit	13
42	inside	10.20.5.21	any	ip	permit	16
45	inside	10.20.6.1-11	any	ip	permit	19
55	inside	10.20.7.20	any	ip	permit	28
57	inside	10.20.6.30	any	ip	permit	30
71	outside	any	any	icmp	permit	40
81	outside	10.20.4.40	any	icmp	permit	50
186	outside	100.200.50.60	100.250.10.10-20	tcp/smtp	deny	112
187	outside	any	100.250.10.10-20	tcp/smtp	permit	113

Como visto anteriormente, as regras passam, inicialmente, por um processo de simplificação. As colunas ”ID Pre-Simp”e ”ID Pos-Simp”indicam, respectivamente, o número (prioridade) da regra em questão antes e depois do processo de simplificação. Por exemplo, a primeira regra da lista encontrava-se a posição 10 antes da simplificação e encontra-se na posição 18 após a simplificação. Por questões de confidenciabilidade, não exibimos os IPs reais utilizados na universidade e trocamos seus números reais, com o cuidado de não alterar suas relações.

6. Conclusões e trabalhos futuros

Este trabalho realizou uma análise do algoritmo de detecção de inconsistências proposto por [Al-Shaer and Hamed 2004]. Para tal, implementamos o algoritmo fornecido em pseudo-código e verificamos sua validade utilizando o *firewall* de exemplo, sendo ambos fornecidos por [Al-Shaer and Hamed 2004].

Discrepâncias entre os resultados do algoritmo e os resultados exibidos foram encontradas. Assim, implementamos nosso próprio algoritmo na ferramenta *open source* DETOX, de maneira a incluir adaptações para corrigir tais discrepâncias e o validamos utilizando o mesmo *firewall* de exemplo usado por [Al-Shaer and Hamed 2004] chegando assim aos resultados esperados.

A viabilidade do novo algoritmo foi testada em uma *firewall* real, onde 307 regras simples foram analisadas e inconsistências foram detectadas, com todo o processo ocor-

rendo em um tempo abaixo de 300ms. Os resultados foram devidamente apresentados ao administrador de rede responsável.

Como trabalhos futuros, pretendemos modificar o DETOX para lidar com múltiplos *firewalls*, bem como adaptá-lo para ser aplicado em redes definidas por *software* (SDN), sobre protocolos como *OpenFlow* e *Border Gateway Protocol* (BGP).

Mais ainda, inspirados pelo trabalho de [Yuan et al. 2006], pretendemos trazer para nossa ferramenta a identificação de quaisquer tipos de inconsistências que possam existir entre combinações de n regras, não somente para *firewalls*, mas também para os ambientes citados acima. Atualmente cada par é analisado de maneira independente pela ferramenta, mas, no caso de um conjunto de regras gerar uma inconsistência com uma única regra ou um outro conjunto de regras, cada um dos pares possíveis desses conjuntos é reportado como um caso de inconsistência. Tal análise pode gerar indicativos de inconsistências ainda mais elaborados e planeja-se incorporá-la ao DETOX em breve.

Além disso, há a possibilidade de adaptar nossa ferramenta para funcionamento *on-the-fly*, constantemente observando as regras existentes no ambiente a ser analisado. Acreditamos que, para redes SDN o maior desafio será a viabilidade devido ao grande volume de mudanças ocorrendo nas redes.

Referências

Al-Shaer, E. and Hamed, H. (2004). Modeling and management of firewall policies. *Network and Service Management, IEEE Transactions on*, 1(1):2–10.

Bartal, Y., Mayer, A., Nissim, K., and Wool, A. (1999). Firmato: a novel firewall management toolkit. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 17–31.

Eppstein, D. and Muthukrishnan, S. (2001). Internet packet filter management and rectangle geometry. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 827–835, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Hari, A., Suri, S., and Parulkar, G. (2000). Detecting and resolving packet filter conflicts. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1203–1212 vol.3.

Khorchani, B., Halle, S., and Villemaire, R. (2012). Firewall anomaly detection with a model checker for visibility logic. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 466–469.

Mayer, A., Wool, A., and Ziskind, E. (2000). Fang: a firewall analysis engine. In *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pages 177–187.

Mukkapati, N. and Ch.V.Bhargavi (2013). Detecting policy anomalies in firewalls by relational algebra and raining 2d-box model. *International Journal of Computer Science and Network Security, IJCSNS*, 13(5).

Yuan, L., Chen, H., Mai, J., Chuah, C.-N., Su, Z., and Mohapatra, P. (2006). Fireman: a toolkit for firewall modeling and analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15 pp.–213.