PolicyViewer: Ferramenta para Visualização e Análise de Políticas de Seguranças em Grafos

Diogo Ditzel Kropiwiec[1]*, **Paulo Lício de Geus**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6.176 – 13.084-971 – Campinas – SP – Brasil

Abstract. *This paper presents the* ***PolicyViewer*** *tool. This tool has been designed to assist administrators with security policies configuration and management, improving policy comprehension, analysis and validation. To achieve these goals, the SELinux security policy model has been mapped in a graph structure, and this structure was used to create visual representations to enhance policy rules understanding and analysis.*

Resumo. *Neste artigo é apresentada a ferramenta* ***PolicyViewer****, cujo objetivo é auxiliar na configuração e gerenciamento de políticas de segurança, auxiliando o administrador na compreensão, análise e verificação das políticas. Para atingir esse objetivo, o modelo de política de segurança do SELinux foi mapeado para uma estrutura de grafo, a partir do qual foram desenvolvidas representações visuais com intuito de permitir uma melhor compreensão e análise das regras da política.*

1. Introdução

O uso cada vez mais difundido de computadores nas mais diversas áreas de atuação, aliado à interligação global provida pela Internet trouxe inúmeros benefícios às organizações e aos usuários de computadores. Entretanto, essa difusão cada vez maior do uso de redes de computadores trouxe consigo uma crescente preocupação com relação à segurança das informações geridas por eles, visto que invasões aos sistemas de informação podem partir de qualquer ponto do mundo, aumentando significativamente o número de atacantes potenciais à rede de uma organização [Nakamura and de Geus 2002].

Para garantir a segurança de sistemas computacionais, os esforços têm sido direcionados a três focos principais de ação: segurança de redes, através do uso de criptografia, *firewalls* e detectores de intrusão [Garfinkel and Spafford 1996]; segurança de aplicações, através do uso de técnicas de programação segura [Viega and McGraw 2003] e mecanismos de detecção de falhas [Cowan et al. 1998] (em especial, *buffer overflow*); e segurança de sistemas operacionais, sem a qual não é possível garantir a segurança dos dois anteriores [Loscocco et al. 1998].

Dos três focos apresentados anteriormente, o mais difícil de se implementar é a segurança de sistemas operacionais, pelo fato de que os sistemas operacionais mais usados ao redor do mundo são derivados de sistemas operacionais anteriores à Internet, e por isso os mecanismos de segurança implementados não são capazes de conter adequadamente a invasão do sistema operacional [Loscocco et al. 1998]. Infelizmente, a

*Apoio CNPq.

adoção de sistemas operacionais seguros por organizações esbarra em diversas questões de caráter financeiro e técnico, entre as quais destacam-se o custo envolvido na adaptação das aplicações usadas aos novos sistemas operacionais, e as dificuldades adicionais impostas na configuração e gerenciamento dos aspectos de segurança do sistema operacional [Kropiwiec and de Geus 2004].

Tendo em vista a questão relacionada à configuração e gerenciamento dos aspectos de segurança, em especial no que tange a compreensão, verificação e validação de políticas de segurança, foi desenvolvida uma ferramenta para mapear a estrutura de políticas de segurança numa representação em grafo, denominada **PolicyViewer**. Esta ferramenta tem por objetivo permitir a visualização das inter-relações entre as diversas regras de maneira clara pelo usuário e a validação automática de regras a partir de características que possam ser obtidas da estrutura de grafo. O **PolicyViewer** foi desenvolvido para o sistema operacional SELinux, devido a sua compatibilidade com o Linux, o que reduziria o custo envolvido na migração de aplicações ao novo sistema.

Na Seção 2 são apresentados os conceitos gerais de políticas de segurança, explicando em maiores detalhes o modelo de políticas do SELinux. Na Seção 3 são apresentadas as características principais da ferramenta **PolicyViewer**, detalhando a forma de mapeamento da política, as representações usadas e a interface com os usuários. Em seguida, são analisados trabalhos correlatos na Seção 4 e um caso de uso da ferramenta na seção 5. A seção 6 apresenta as conclusões finais e possíveis extensões do projeto.

2. Políticas de Segurança

Os métodos de garantir segurança a um sistema operacional e às informações contidas nele podem ser agrupadas em: mecanismos de autenticação de usuário, mecanismos de controle de acesso e mecanismos de auditoria. Destes três, o interesse maior para este artigo recai sobre o segundo grupo: mecanismos de controle de acesso. De um modo geral, é relativamente simples para o sistema operacional representar as permissões de acesso de usuários sobre objetos do sistema, utilizando uma matriz de controle de acessos. A forma de armazenamento da matriz pode mudar de um sistema para outro, através do uso de *ACLs* (*Access Control Lists*) associadas aos objetos ou do uso de *Capabilities* associadas aos sujeitos (usuários e processos) do sistema operacional. A granularidade do controle do acesso também pode variar, dependendo dos objetivos de segurança desejados para o sistema [Bishop 2003].

Entretanto, essa forma direta e eficiente para o sistema operacional representar as permissões de controle de acesso não é prática na administração da segurança de sistema. Por isso, no intuito de simplificar a especificação dos critérios de segurança, e torná-los mais claros ao usuário, foram desenvolvidos modelos mais abstratos, denominados políticas de segurança. Essas abstrações utilizam estruturas baseadas em características do sistema operacional (tipos de objetos, usuários, processos, domínios de acesso) ou em características externas ao sistema (hierarquia, cargo ou função em uma organização, finalidade do objeto) para definir de forma mais condensada, e em alguns casos hierárquica, o relacionamento entre sujeitos e objetos do sistema.

Infelizmente, não existe um modelo de política de segurança que possa ser considerado completo. Cada organização e ambiente computacional possui objetivos próprios de segurança, de forma que a complexidade necessária a uma política de segurança

completa seria um empecilho para seu uso. A grande variabilidade de requisitos tornou necessário desenvolver políticas que privilegiassem apenas determinados critérios de segurança, de forma que elas são classificadas em função das premissas básicas de segurança ou de seu uso mais clássico. Assim, as políticas são agrupadas em *políticas de confidencialidade*, *políticas de integridade* e *políticas híbridas*, onde o quesito refere-se aos objetivos de segurança de cada grupo (ou seja, confidencialidade, integridade, ou um misto de ambos), ou são agrupadas em políticas *militares* (ou *governamentais*) e *comerciais*, em função do foco para o qual foram criadas.

Um dos modelos de políticas mais importantes é o *Modelo de Bell-LaPadula* [Bell and La Padula 1973], por ter sido o primeiro a definir segurança utilizando uma notação matemática completa e ter influenciado o desenvolvimento de outros modelos e de tecnologias de segurança de computadores. O modelo tratava exclusivamente de confidencialidade, porém em [Biba 1977] é apresentado o dual matemático da política para tratar aspectos de integridade de informação. Posteriormente, [Lipner 1982] apresentou uma combinação dos dois modelos, mais adequada aos requisitos comerciais.

Por se restringirem a apenas um aspecto de segurança (confidencialidade ou integridade), e por dependerem fortemente de notação matemática, esses modelos impõem maior complexidade à especificação de regras de segurança, sendo que outros modelos (híbridos) são mais comumente implementados em sistemas operacionais. Desses modelos, serão tratados o *DTE* (*Domain and Type Enforcement*) e o *RBAC* (*Role based access control*), visto que esses são os modelos utilizados para especificação de políticas de segurança no SELinux.

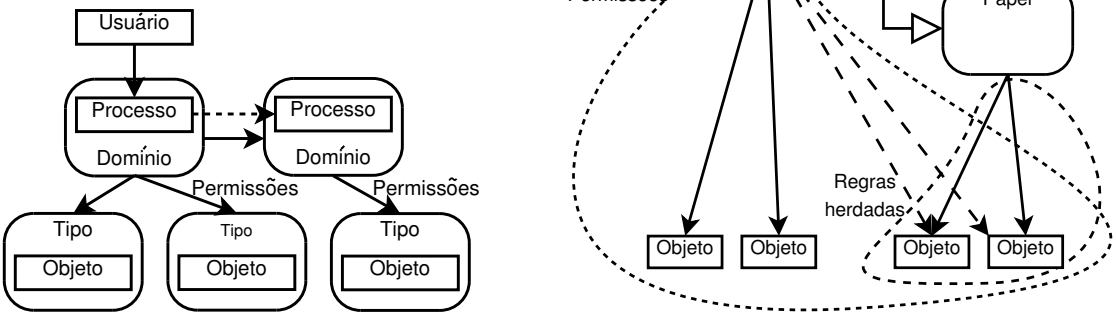
2.1. Modelo de Política do SELinux

O SELinux (*Security-Enhanced Linux*) [Loscocco and Smalley 2001] consiste na implementação da arquitetura de segurança *Flask* [Spencer et al. 1999] ao núcleo (*kernel*) do sistema operacional Linux, de forma a incorporar um mecanismo de controle de acesso mandatório forte e flexível e prover mecanismos de isolamento de informações baseados tanto em requisitos de confidencialidade quanto integridade. Por manter a compatibilidade com as aplicações já existentes para o Linux, o SELinux reduz consideravelmente o custo de migração para um sistema operacional seguro, e tem grandes chances de ser cada vez mais utilizado que outros sistemas operacionais seguros existentes.

O SELinux permite a implementação de uma grande variedade de políticas de segurança, graças à característica de flexibilidade herdada da arquitetura *Flask*. Como o modelo base disponibilizado junto com o SELinux compreende o uso conjugado do *DTE* e do *RBAC*, estes dois foram escolhidos como base para o desenvolvimento da ferramenta **PolicyViewer**.

O modelo de *DTE* é derivado do modelo de *TE* (*Type Enforcement*) desenvolvido por Boebert e objetiva restringir o acesso de processos a objetos do sistema operacional a partir de uma política de segurança baseada em tipos e domínios [Walker et al. 1996]. Nesse modelo, todos os objetos do sistema estão associados a um *tipo*, que representa uma classe de equivalência de dados e aplicações (por exemplo, arquivos do setor de recursos humanos, manufatura, vendas) que são tratados da mesma forma pela política. Paralelamente, todo processo em execução está associado a um único *domínio*, que determina quais as ações permitidas ao processo quando acessando objetos de tipos específicos ou

interagindo com outros domínios (Figura 1(a)).



(a) Esquema geral da política *DTE*

(b) Esquema geral da política *RBAC*

Figura 1. Comparação entre as políticas *DTE* e *RBAC*

Para especificação das regras da política foi definida uma linguagem de especificação, o *DTEL* (*DTE Language*), que utiliza uma combinação de elementos de baixo e alto nível para expressar as restrições de domínios a tipos. Por apresentar essas características e não utilizar um modelo matemático em sua definição, o *DTEL* é classificado como uma linguagem de alto nível. Com ela, é possível definir a segurança do sistema operacional em função de seus elementos (usuários, processos, arquivos e recursos do sistema) em um nível de granularidade que pode variar de implementação para implementação (por exemplo, considerar somente leitura, escrita e execução, ou então considerar operações mais finais—escrita ao final, criação de arquivos e/ou diretório).

O *RBAC* [Ferraiolo and Kuhn 1992, Ferraiolo et al. 1995], por sua vez, trata o acesso não em função do usuário em si, mas sim em função do seu papel (*Role*) na organização em que está inserido. Embora tanto o *DTE* quando o *RBAC* possam especificar o mesmo tipo de segurança em um modelo estático, o segundo é mais adequado para tratar situações de mudança de atividades de funcionários da organização. Por exemplo, considere o caso de um arquivista da organização que é promovido para o setor de recursos humanos da empresa. Após a promoção, as permissões de acesso ao sistema devem ser modificadas para condizer com sua nova posição. De forma similar, quem o substituir deverá ter o mesmo acesso que ele possuía. Repare que, nesse caso, é o cargo do usuário, e não sua identidade, que influencia as decisões de segurança.

A base do *RBAC* é, portanto, o conceito de *papel*, que corresponde ao direito de realizar um conjunto de transações relacionadas a uma atividade específica. Diferente do conceito de domínio no *DTE*, os papéis estão associados aos usuários do sistema operacional, e não aos processos, de tal forma que um usuário pode estar autorizado a atuar em mais de um papel. Outra característica do *RBAC*, derivada do ambiente de organizações, é a hierarquia de papéis, que permite a inclusão de todas as permissões referentes a um determinado papel (denominado sub-papel) em outro papel. Dessa forma, evita-se redundâncias na especificação da política, e ocorrendo a modificação das permissões refe-

rentes a um sub-papel, todos os papéis que o incluem também estarão condizentes com a mudança da política (Figura 1(b)).

Um detalhe da especificação do *RBAC* é que ele define apenas as regras de relacionamento entre papéis, ficando a cargo da implementação determinar como serão representadas as permissões sobre objetos e como estas são gerenciadas. O SELinux aproveita esse gancho para associar os domínios do modelo *DTE* aos papéis do modelo *RBAC*, de tal forma que cada papel está autorizado a utilizar um ou mais domínios, e as regras de transição de domínios do *DTE* são limitadas pelo papel associado ao processo ou usuário realizando a transição.

Outro adendo implementado pelo SELinux, no intuito de simplificar a especificação das regras de política de segurança, é o conceito de classes de objetos. A *classe* do objeto define como o objeto é tratado pelo sistema, e quais as permissões disponíveis para configuração sobre esse objeto, de forma a distinguir dois objetos do mesmo tipo tratados diferentemente pelo sistema. Por exemplo, um tipo genérico `user_data` pode estar associado tanto aos arquivos quanto aos diretórios do usuário, que possuem conjuntos de permissões diferentes. A granularidade implementada no SELinux permite distinguir objetos em sub-classes, como arquivos de *links* ou de dispositivo. Com o uso de classes, o SELinux provê permissões em granularidade mais fina do que o tipicamente esperado para o *DTE*.

3. PolicyViewer

Embora as políticas de segurança simplifiquem grandemente a especificação dos critérios de segurança do sistema operacional, a complexidade da interação do sistema operacional com os mais diversos arquivos e dispositivos, em especial da parte relativa ao acesso a arquivos e recursos privilegiados, torna a política complexa e extensa. O fato das políticas serem especificadas em formato texto impõe uma estrutura linear às regras que apenas dificulta ainda mais sua análise e verificação. Apenas para exemplificar, o arquivo de especificação da política do SELinux para definir os serviços básicos do sistema possui mais de 7.700[1] linhas de especificação, não contando comentários e linhas em branco.

Pelos dados acima, observa-se que identificar e corrigir erros na política é um processo complicado, e que a adição ou modificação de regras que poderiam causar algum conflito com outros trechos da especificação é difícil de ser verificada e validada. Com o intuito de quebrar a linearidade imposta pelo formato textual foi desenvolvida a ferramenta **PolicyViewer**, que tem por objetivo representar as regras da política de segurança utilizando uma estrutura de grafos, e permitir a visualização dessa estrutura pelo usuário.

Serão apresentadas a seguir as principais características do **PolicyViewer**, envolvendo o mapeamento da política em grafos, suas representações visuais, a interface disponível para o usuário interagir com o sistema, e uma breve discussão sobre o ambiente de desenvolvimento e arquitetura adotada. Mais detalhes sobre a ferramenta podem ser obtidos em [Kropiwiec 2005].

3.1. Políticas de Segurança em Grafo

A primeira etapa do desenvolvimento consistiu em mapear as regras da política do SELinux em uma estrutura de grafos, que permitisse analisar as regras conforme o relacio-

[1] O desenvolvimento foi baseado na versão 1.17 da política de segurança

namento dos vários elementos envolvidos na especificação: usuários, papéis, domínios, tipos e as regras de transição e acessibilidade. Além de permitir melhor visualização da estrutura especificada pela política, obtém-se a vantagem de poder usar, a partir dessa representação, algoritmos de grafos para auxiliar na verificação e validação, identificando pontos que poderiam implicar em falhas na segurança e alertar o administrador para verificar com maior atenção às regras envolvidas.

A idéia básica consiste em mapear os elementos da política (tipos, domínios, papéis e usuários) em nós e as regras (transição e acessibilidade) em arestas orientadas, armazenando as informações necessárias para distinguir nós e arestas com significados distintos e permitir identificar as linhas na especificação que correspondem à regra representada. Dessa forma, poderão ser analisadas as relações e as implicações das regras no comportamento do sistema operacional, permitindo identificar eventuais pontos de vulnerabilidade.

O mapeamento, entretanto, não irá cobrir todas as definições presentes na política, restringindo-se apenas às regras que influenciam no comportamento do sistema. O conjunto de regras da política do SELinux consiste de um conjunto de definições que podem ser agrupadas nos seguintes conjuntos: *Flask*, *DTE*, *RBAC*, usuários, contexto de segurança e restrições [Smalley 2003]. Destes conjuntos, dois não são mapeados em grafos por não estarem relacionados ao comportamento da política. O conjunto *Flask* consiste de definições de estruturas compartilhadas entre o núcleo do SELinux e a política de segurança, que raramente sofrem modificações, e o conjunto de contextos de segurança define quais as associações entre objetos do sistema com os tipos da política.

As definições *DTE* especificam os nomes dos tipos e seus atributos, as regras de transições entre domínios e entre domínios e tipos, e as regras de acessibilidade, auditoria e asserções. Um detalhe importante a ser notado com relação às definições *DTE* do SELinux é que ele trata domínios como tipos especiais, permitindo a utilização de uma estrutura uniforme para representação das regras entre domínios e entre domínios e tipos. Dessa forma, tanto domínios quanto tipos são representados por um mesmo tipo de nó, denominado `TypeNode`. As regras de acessibilidade são definidas por arestas `TERuleEdge`, orientadas do domínio para o tipo ao qual a regra se aplica. Por sua vez, as regras de transição entre domínios e tipos e entre domínios são orientadas do domínio origem para o domínio (ou tipo) destino, sendo representadas por arestas `TETransitionEdge`. Como normalmente a transição envolve um tipo intermediário — o tipo da aplicação a ser executada ou do objeto *container* em que o novo objeto será criado — essa informação é armazenada na aresta, de forma a permitir identificar arestas que poderiam ser exploradas caso o usuário pudesse criar aplicações do tipo intermediário. As asserções não são mapeadas visto que implicam na inexistência de arestas no mapeamento previsto.

As definições *RBAC* são utilizadas para definir os papéis e os domínios a que estão autorizados, hierarquia de papéis e regras de acessibilidade entre papéis. Diferentemente das regras do *DTE*, que utilizam regras próprias para restringir as transições entre domínios, as transições entre papéis são limitadas pelo conjunto de restrições dinâmicas, que impõe uma estrutura mais complexa ao mapeamento em grafo das regras *RBAC* e, portanto, não são mapeadas no grafo. Os papéis são representados por nós `RoleNode`, as autorizações sobre domínios são armazenadas em arestas `RoleTypeEdge`, orientadas

do papel para o tipo, e as regras de transição entre papéis são representadas por arestas `RBACTransitionEdge`.

Finalmente as declarações de usuários definem os usuários reconhecidos pela política e os papéis a que estão autorizados, fechando a especificação do comportamento geral do sistema. Os usuários são representados por nós `UserNode`, enquanto as autorizações sobre papéis são representadas por arestas `UserRoleEdge`.

A partir dessa representação, algoritmos aplicados a grafos podem ser aplicados, com pequenas modificações, sobre a estrutura de grafo para obter informações sobre a política. Por exemplo, um algoritmo de percurso em grafo pode ser utilizado para determinar quais os tipos acessíveis (e com que permissões) a partir de um domínio, papel ou usuário. Seguindo as arestas e revertendo sua orientação, é possível determinar a quem estão expostos determinados tipos. Outras questões, como exclusão mútua de papéis, podem ser verificadas analisando a conexidade entre nós específicos do grafo. Para este caso, por exemplo, a existência de um duplo caminho entre um usuário e um tipo restrito, através de papéis mutuamente exclusivos, é um forte indicativo de um ponto de falha na política. Finalmente, outra questão importante na segurança é a identificação e eliminação de fluxos de informação indiretos, que poderiam quebrar as premissas de segurança do sistema. Para tal, basta inverter a orientação das arestas de acesso de escrita, e aplicar o algoritmo de percurso em grafos para determinar se determinado usuário (ou papel) pode obter acesso a uma informação "segura" de forma indireta.

3.2. Representação Visual

Com o mapeamento em uma estrutura de grafo da política de segurança, foram obtidas algumas propriedades interessantes para sua verificação automática. Entretanto, mais do que simplesmente a automatização do processo de verificação, a ferramenta tem por objetivo auxiliar na compreensão da política de segurança, através do uso de representações gráficas baseadas em grafo. Nessa seção, serão apresentadas as representações desenvolvidas para exibir, de maneira clara e independente da linguagem de especificação, o relacionamento entre os elementos da política.

Para facilitar a visualização, utilizou-se cores diferentes para nós representando elementos diferentes da política (usuário, papel, domínio e tipo) e arestas com desenho diferente para representar "ações" diferentes. A Figura 2 apresenta o diagrama de acessibilidade do **PolicyViewer**, que corresponde diretamente ao mapeamento em grafo da política de segurança do SELinux. Estão destacadas na figura as arestas de autorização de acesso a papéis e domínios (`a`, `c` e `d`), as arestas de transição entre papéis, entre domínios, e de tipos (`b`, `e` e `f`) e as arestas de permissão (`g`). Repare que nas arestas de transição entre domínios (e) e de tipo (f), são representados o tipo da aplicação usada na transição de domínio e o tipo do objeto *container* onde o novo objeto será criado.

Através desse diagrama, é possível verificar os "domínios" (no sentido amplo) de acesso de um determinado usuário, papel ou domínio. Acessos a tipos restritos podem ser verificados pela existência ou não de arestas ligando domínios-chave a esses tipos. Conflitos de interesse entre papéis podem ser determinados através da presença de caminhos duplos para elementos passíveis de exclusão mútua, além de diversos outros aspectos de segurança interessantes para análise.

Entretanto, esse diagrama não permite, por exemplo, a identificação de caminhos

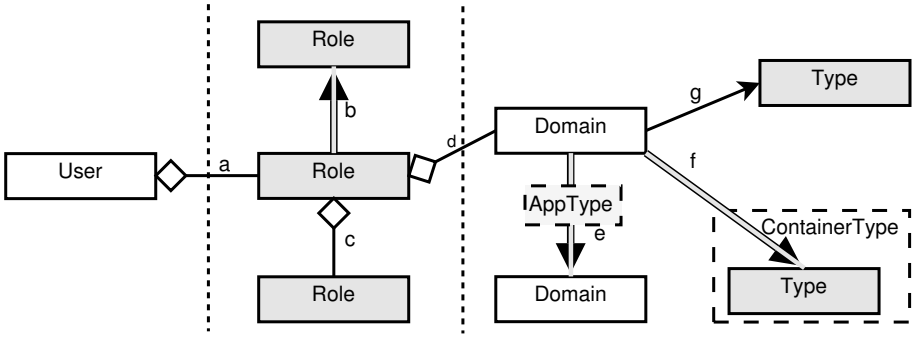


Figura 2. Representação Visual do diagrama de acessibilidade do PolicyViewer, com destaque para a diferenciação entre nós representando elementos diferentes e para arestas com significados diferentes.

de fluxo de informação que possam ocasionar vazamento de informações sigilosas. Por isso, desenvolveu-se uma segunda representação, similar a primeira, que permitisse a identificação de tais caminhos. Nessa nova representação, as arestas de acessibilidade (arestas `g` na Figura 2) são modificadas, de tal forma que: arestas que representem permissão de leitura são orientadas do tipo para o domínio; arestas que representem permissão de escrita são orientadas do domínio para o tipo, podendo ser bidirecionais caso a aresta seja de permissão de leitura e escrita; e as demais permissões são suprimidas do grafo. A Figura 3 apresenta esse diagrama, denominado diagrama de fluxo de informações, onde as arestas `a`, `b` e `c`, representam, respectivamente, permissões de leitura, escrita, e leitura e escrita.

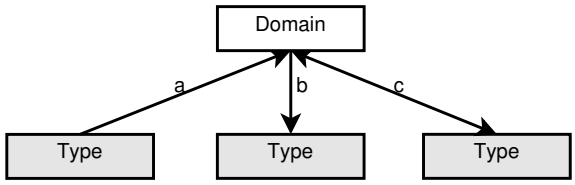


Figura 3. Representação Visual do diagrama de fluxo de informação.

É importante ressaltar que essa representação não permite a identificação de canais cobertos (*covert channels*), uma vez que a política do SELinux não trata dessa questão. Esse problema e questões relacionadas são tratadas no Capítulo 8 de [Bishop 2003].

3.3. Interface

Definido o mapeamento e as representações a serem disponibilizadas pela ferramenta, iniciou-se o projeto da interface a ser utilizada pelo usuário. Como um dos intuitos da ferramenta consiste em auxiliar o usuário na compreensão e visualização das políticas, foram estudadas técnicas de visualização de informação para criar uma interface clara e objetiva, porém provendo o maior conjunto de recursos possíveis para auxiliar o usuário na visualização e análise da política.

Ao se projetar a interface, observou-se a necessidade de uma exibição parcial do grafo, visto que o grafo completo representando a política toda não iria contribuir para a compreensão do usuário, dado o número de nós e arestas resultantes do mapeamento. Por isso, optou-se pela representação localizada de aspectos da política, obtida através do

uso de filtragem de informações ou aplicação de algoritmos para geração de subgrafos. Por exemplo, pode-se apresentar apenas os relacionamentos *DTE* acessíveis por um determinado papel da política (para verificar se o papel está corretamente especificado), ou então optar pela exibição dos caminhos de transição que interligam dois domínios (para descobrir eventuais transições perigosas à segurança do sistema).

Outro aspecto observado durante a prototipação foi a necessidade de utilizar algoritmos que realizassem a distribuição dos nós de maneira coerente e exibí-los de forma a não poluir a representação com detalhes desnecessários, e, adicionalmente, permitir interação do usuário com o posicionamento dos nós. Dessa forma, o usuário poderia focar nos aspectos de seu interesse. Diversas técnicas foram estudadas para compor a ferramenta, como *fish-eye view*, *force directed graphs* [Card et al. 1999], entre outras.

A Figura 4 apresenta a interface principal do **PolicyViewer**, com um exemplo de visualização parcial da política. Após abrir o arquivo de política a ser analisado, o usuário tem a sua disposição um conjunto de funções para selecionar o que deseja visualizar. Na figura, o trecho da política apresentado corresponde aos domínios acessíveis ao domínio `user_t`. A partir desse ponto, o usuário pode optar por esconder nós ou expandir as permissões de acesso de um determinado domínio, de forma a percorrer a política em busca de características específicas não obtidas automaticamente pelos algoritmos disponibilizados.

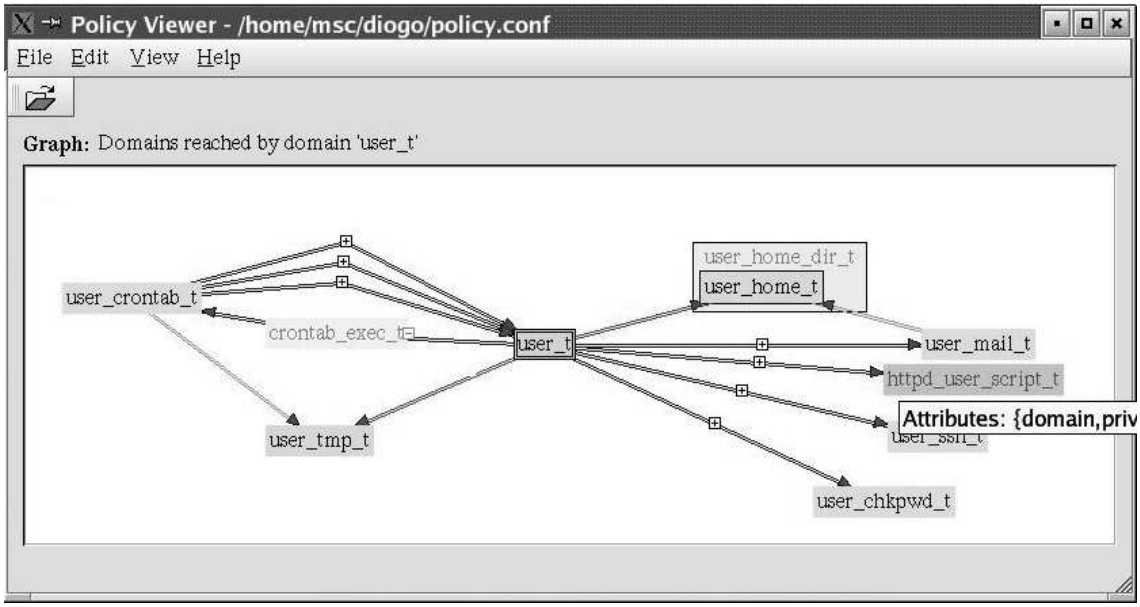


Figura 4. Interface principal com o usuário da ferramenta PolicyViewer.

Como pode ser observado na Figura 4, são utilizadas *hints* para apresentar algumas informações adicionais sobre um determinado nó ou aresta representado (no exemplo, os atributos de um tipo). Entretanto, como o objetivo é auxiliar na análise da política, mais informações são necessárias, de forma que o usuário possa optar por ver o detalhamento completo, incluindo informações como arquivo e linha da especificação correspondente ao nó ou aresta em questão (Figura 5).

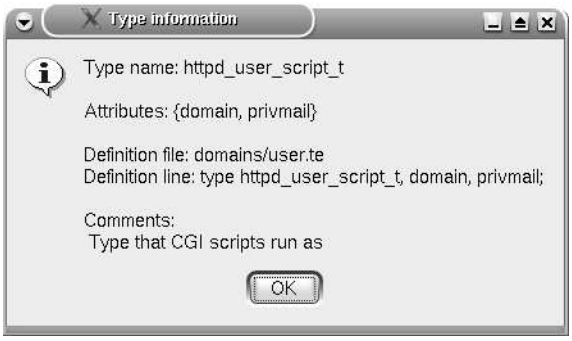


Figura 5. Janela de detalhes de um nó "tipo"

4. Trabalhos Correlatos

Em [Koch et al. 2002] é apresentada uma formalização do modelo *RBAC* utilizando grafos e transformações em grafos. Essa formalização permite o tratamento uniforme de papéis tanto de usuários quanto administradores, evitando o uso de meta-modelos para descrever possíveis evoluções da estrutura administrativa, beneficiando-se de resultados bem estabelecidos em sistemas de transformações em grafo. Com isso, foi obtida uma técnica que permite a descrição visual e intuitiva de estruturas dinâmicas, útil na validação de políticas construídas com o modelo. Entretanto, o foco do trabalho é puramente voltado para características automáticas extraídas do grafo, não incluindo nenhum mecanismo gráfico de exibição da estrutura criada.

Outro trabalho similar é apresentado em [Nyanchama and Osborn 1999], onde é proposto um modelo de grafos para mapear papéis em hierarquias, utilizadas na representação de relacionamento entre papéis. Utilizando o modelo proposto, é apresentada uma taxonomia para vários tipos de conflitos, considerando detalhes de conflitos entre privilégios e/ou papéis, sendo demonstrado seu uso para particionar o grafo gerado em coleções não-conflitivas que podem ser seguramente autorizadas a um determinado usuário. Com isso, critérios de exclusão mútua podem ser validados.

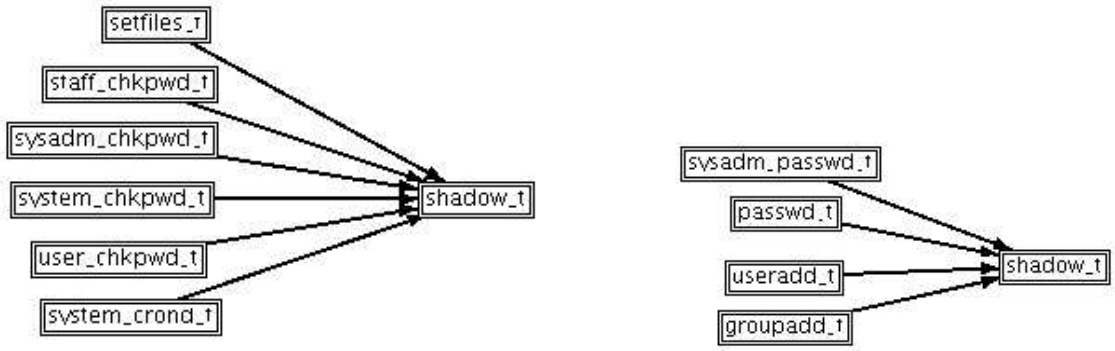
Enquanto nenhum dos dois trabalhos anteriores previa uma ferramenta gráfica, em [Damianou et al. 2002] é apresentado um conjunto de ferramentas integradas, entre as quais está incluso o *Domain Browser*, que provê uma interface com o usuário para gerenciamento de políticas e exibe a política em árvores representando a relação entre domínios e objetos. Entretanto, esse conjunto de ferramentas é direcionado para sistemas distribuídos, utilizando uma linguagem de especificação de políticas própria, denominada Ponder. A ferramenta leva em conta questões de usabilidade para auxiliar o usuário a navegar entre os domínios do sistema, incluindo menus de contexto e integração com outras ferramentas, utilizando o algoritmo de árvores hiperbólicas [Card et al. 1999] para disposição dos nós da árvore.

Não foi encontrado nenhum trabalho nos moldes da ferramenta apresentada neste artigo, baseado em segurança de sistemas operacionais, de forma que não foi possível estabelecer critérios de comparação adequados. A grande vantagem com relação aos trabalhos citados é que a ferramenta **PolicyViewer** se propõe a tratar a política como um todo, e não se restringir a aspectos específicos da política para análise e visualização.

5. Caso de Uso: Exposição do Arquivo de Senhas

Para demonstrar o funcionamento da ferramenta, foi escolhido um caso de uso referente a uma importante questão de segurança de sistemas operacionais. A exposição do arquivo de senhas do sistema aos usuários pode implicar em graves riscos à segurança. Havendo permissão de escrita, um usuário poderia alterar a senha de outro usuário (inclusive administradores do sistema) de formar a lhe garantir acesso irrestrito ao sistema. Menos grave, mas também igualmente importante para segurança é a possibilidade de algum usuário poder ler o arquivo de senhas. Mesmo com o uso de funções de *hash* para obscurecer as senhas, existem métodos de ataque de dicionário que podem ser utilizados para obter senhas de outros usuários.

Como base para análise, foi escolhida a política padrão do SELinux disponibilizada em conjunto com a distribuição Gentoo Linux. A primeira etapa da análise consiste em descobrir o tipo associado aos arquivos `/etc/passwd` e `/etc/shadow`, que correspondem aos registros dos usuários e suas senhas. No caso, o tipo associado é `shadow_t`. Utilizando-se o **PolicyViewer**, primeiramente são identificados os domínios que possuem permissão de leitura e/ou escrita sobre estes arquivos (Figura 6).



(a) Acesso somente de leitura

(b) Acesso de escrita

Figura 6. Domínios com acesso ao tipo *shadow_t*

Por questões de síntese, será apresentado apenas o trecho da análise envolvendo as permissões de escrita. Determinando os domínios que possuem acesso, a etapa seguinte consiste em identificar quais os caminhos de transições sucessivas que podem levar aos domínios `passwd_t`, `sysadm_passwd_t`, `useradd_t` e `groupadd_t`. Utilizando o **PolicyViewer**, foi obtido o diagrama apresentado na Figura 7, que além de apresentar as transições, permite visualizar quais os tipos das aplicações envolvidas na transição (no exemplo, destacou-se apenas o tipo da aplicação para transição do domínio do usuário comum). Repare que poucos domínios permitem o acesso indireto ao arquivo de senhas e que, ignorando-se o acesso a partir do administrador do sistema, o único domínio acessível é o `passwd_t` e que a transição ocorre através de aplicações do tipo `passwd_exec_t`. Analisando finalmente as aplicações associadas, descobre-se que apenas a aplicação `/usr/bin/passwd` está associada a este tipo, do que se conclui que, na ausência de falhas nessa aplicação, um atacante que obtenha acesso à conta de um usuário comum não poderá comprometer o arquivo de senhas.

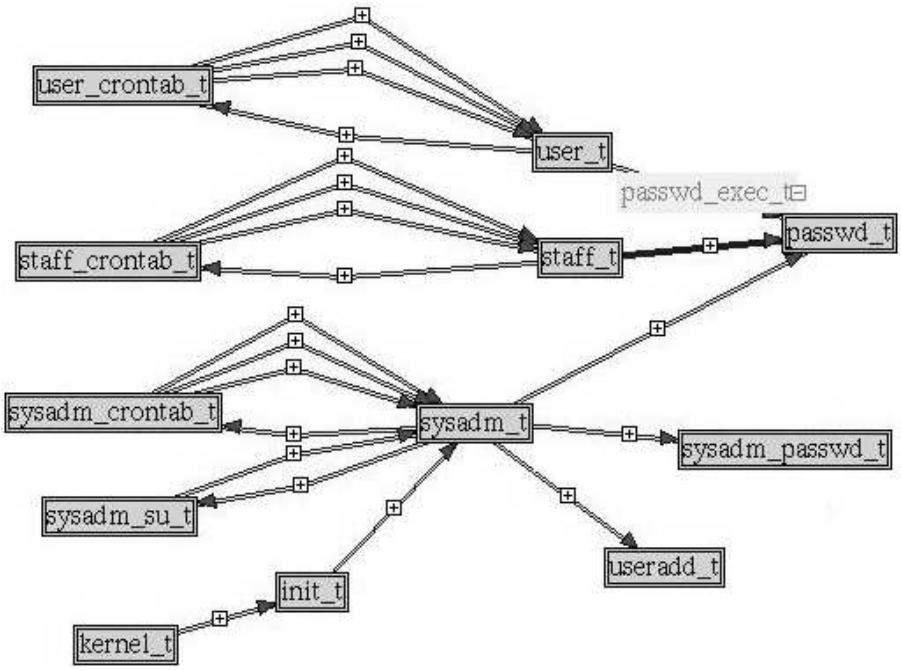


Figura 7. Janela de detalhes de um nó "tipo"

Tentar realizar a mesma análise utilizando o arquivo de definições de política em seu formato texto é impraticável. Em [Kropiwiec 2005] é apresentada essa tentativa, e outros casos de uso da ferramenta.

6. Conclusão

O surgimento e evolução da Internet trouxeram inúmeros benefícios a organizações e usuários de computadores, mas também expôs os computadores e seus sistemas a ataques provenientes do mundo todo. Infelizmente, os sistemas operacionais utilizados são derivados de sistemas anteriores a Internet, e por isso sistemas operacionais seguros precisam ser adotados. Entretanto, a adoção desses sistemas operacionais ainda é lenta, devido a questões envolvendo o custo de migração e a complexidade de gerenciar as políticas de segurança.

Neste artigo, foi apresentada a ferramenta **PolicyViewer**, cujo objetivo é auxiliar na visualização e análise de políticas de seguranças do SELinux utilizando a estrutura de grafos. Primeiramente, foram analisados os modelos de política de segurança existentes, com foco no modelo adotado pelo SELinux. Em seguida, foram apresentados os passos envolvidos no projeto e desenvolvimento da ferramenta, com interesse especial no mapeamento da política para a estrutura de grafo e sua representação visual. Finalmente, foram apresentados trabalhos correlatos e um exemplo de uso da ferramenta.

Como pode ser observado, a ferramenta permite uma melhor análise da política de segurança, ao quebrar a linearidade imposta pela especificação em formato textual. Através da navegação pelos usuários, papéis, domínios e tipos da política, é possível identificar diversas características do relacionamento entre as regras, aumentando a compreensão geral do administrador ante a política especificada. Um ponto a ser explorado em trabalhos futuros sobre a ferramenta consiste em fazer um estudo detalhado de propri-

edades de grafo que possam ser pareadas a modelos de segurança, permitindo verificações automáticas da estrutura gerada. Outro aspecto que pode ser estudado é o uso de outras estruturas de representação, além de grafos, para mapeamento e validação do modelo.

Referências

Bell, D. E. and La Padula, L. J. (1973). Secure computer systems: Mathematical foundations and model. Technical Report MTR-2547 Vol I, MITRE Corporation, Bedford, MA, EUA.

Biba, K. (1977). Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, EUA.

Bishop, M. (2003). *Computer Security: Art and Science*. Addison-Wesley, 4th edition.

Card, S. K., Mackinlay, J. D., and Shneierman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, CA, USA.

Cowan, C., Pu, C., Maier, D., Hinton, H., andPeat Bakke, J. W., Beattie, S., Grier, A., Wagle, P., and Zhang, Q. (1998). Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Conference*.

Damianou, N., Dulay, N., Lupu, E., Sloman, M., and Tonouchi, T. (2002). Tools for domain-based policy management of distributed systems. In *In proceedings of Network Operations and Management Symposium, 2002.*, pages 203 – 217.

Ferraiolo, D. and Kuhn, R. (1992). Role-based access control. In *Proceedings of The 15th National Computer Security Conference*.

Ferraiolo, D. F., Cugini, J. A., and Kuhn, D. R. (1995). Role-based access control (rbac): Features and motivations. *Proceedings of the 7th Annual Computer Security Applications Conference*.

Garfinkel, S. and Spafford, G. (1996). *Pratical Unix & Internet Security*. O'Reilly & Associates, Inc., USA, 2nd edition. 971 p.

Koch, M., Mancini, L. V., and Parisi-Presicce, F. (2002). A graph-based formalism for rbac. *ACM Trans. Inf. Syst. Secur.*, 5(3):332–365.

Kropiwiec, D. D. (2005). Policyviewer: Ferramenta para visualização de políticas de segurança em grafos. Master's thesis, Universidade Estadual de Campinas (UNICAMP).

Kropiwiec, D. D. and de Geus, P. L. (2004). Paradigmas de segurança em sistemas operacionais. In *Anais do IV Workshop de Segurança em Sistemas Computacionais*, Gramado/RS.

Lipner, S. (1982). Non-discretionary controls for commercial applications. In *Proceedings of th 1982 Symposium on Privacy and Security*, pages 2–10.

Loscocco, P. and Smalley, S. (2001). Integrating flexible support for security policies into the linux operating system. In *Proccedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, Boston Mass.

Loscocco, P. A., Smalley, S. D., Muckelbauer, P. A., Taylor, R. C., Turner, S. J., and Farrel, J. F. (1998). The inevitability of failure: The flawed assumption of security in modern computing environment. In *Proceedings of the 21st National Information Systems Security Conference*, pages 303–314.

Nakamura, E. and de Geus, P. L. (2002). *Segurança de Redes em ambientes cooperativos*. Editora Berkeley, São Paulo, first edition.

Nyanchama, M. and Osborn, S. (1999). The role graph model and conflict of interest. *ACM Trans. Inf. Syst. Secur.*, 2(1):3–33.

Smalley, S. D. (2003). Configuring the selinux policy. Technical report, National Security Agency of United States of America.

Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Andersen, D., and Lepreau., J. (1999). The flask security architecture: System support for diverse security policies. *Proceedings of the Eighth USENIX Security Symposium*, pages 123–139.

Viega, J. and McGraw, G. (2003). *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 3rd edition.

Walker, K. W., Bagder, D. F., Petkac, M. J., Sherman, L., and Oostendorp, K. A. (1996). Confining root programs with domain and type enforcement. In *Proceedings of The 6th USENIX Security Symposium*, San Jose, California.