Policy Viewer:
Ferramenta para visualização de políticas de segurança em grafos

Diogo Ditzel Kropiwiec

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Policy Viewer: Ferramenta para visualização de políticas de segurança em grafos

Diogo Ditzel Kropiwiec[1]

Fevereiro de 2005

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus (Orientador)
- Prof. Dr. Joni da Silva Fraga
  Departamento de Automação e Sistemas, UFSC
- Prof. Dr. Ricardo Dahab
  Intituto de Computação, UNICAMP
- Prof. Dr. Célio Cardoso Guimarães (Suplente)
  Instituto de Computação, UNICAMP

[1]**O presente trabalho foi realizado com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq - Brasil**

Substitua pela ficha catalográfica

Substitua pela folha com a assinatura da banca

Policy Viewer: Ferramenta para visualização de políticas de segurança em grafos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Diogo Ditzel Kropiwiec e aprovada pela Banca Examinadora.

Campinas, Março de 2005.

Prof. Dr. Paulo Lício de Geus (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Resumo

A Internet trouxe grandes benefícios às organizações e usuários de computadores, porém causou também uma maior exposição dos sistemas computacionais interligados em rede. Inúmeros têm sido os esforços para conter o crescente aumento dos ataques que ocorrem no mundo todo, dentre os quais inclui-se o desenvolvimento de sistemas operacionais mais seguros. Entretanto, a adoção desses sistemas ainda é incipiente, devido a várias dificuldades envolvidas no processo, dentre as quais destaca-se a complexidade de configuração e gerenciamento de políticas de segurança.

Nesta dissertação, são apresentados os aspectos estudados durante o desenvolvimento do mestrado, que permitiram a identificação dos problemas atuais associados a segurança de sistemas operacionais e políticas de segurança. Isso resultou no projeto e implementação do **Policy Viewer**, uma ferramenta de visualização de políticas de segurança. Sua finalidade é auxiliar o administrador de políticas na compreensão, visualização e verificação das políticas de segurança especificadas para o sistema operacional.

Utilizando as características apresentadas no projeto, foi desenvolvida uma implementação parcial da ferramenta contendo um subconjunto das funcionalidades previstas, sobre o qual foram elaborados exemplos para demonstrar sua utilidade no auxílio da configuração de políticas e na identificação de problemas da política especificada.

Abstract

The Internet brought great benefits to organizations and computer users, but has also caused a larger exposure of the computing systems connected to the network. Countless efforts are being made to contain the increasingly higher level of attacks that happen all over the world, among which stands the development of safer operating systems. Unfortunately, the adoption of these systems is still incipient, because of several obstacles involved in the process. One of them is the complexity of configuring and managing security policies.

This dissertation shows aspects of operating system security and security policies studied during the Masters program, leading to the identification of current problems associated with them. This resulted in the project and implementation of **Policy Viewer**, a tool for the visualization of security policies. Its purpose is to aid the policy administrator in the comprehension, visualization and validation of operating systems security policies.

The tool has been partially implemented with a subset of the intended functions, using the features presented in the project. Also, examples are shown to demonstrate its utility toward aiding in the process of policy configuration and in the identification of possible problems of such policies.

"The beginning of knowledge is the discovery of something we do not understand."
"Seek freedom and become captive of your desires. Seek discipline and find your liberty."
Frank Herbert

Agradecimentos

Eu gostaria de agradecer a todos que participaram na minha vida e de alguma forma contribuíram para eu estar onde estou hoje.

A meus pais e irmãos, Vicente, Ignez, Cássio e Celina, agradeço pelo carinho, amor e incentivo, e por tudo que fizeram para garantir meu crescimento, amadurecimento e aprendizado.

A minha namorada, Aretha, pelo seu amor, paciência e compreensão durante todo o mestrado, tendo que suportar a distância imposta pelas minhas atividades na faculdade.

A meu orientador, Paulo, pela orientação, apoio e atenção fornecidas durante o desenvolvimento deste trabalho, sem os quais não conseguiria realizar esse projeto.

A meus tios e tias, primos e primas, avôs e avós, pela preocupação em acompanhar minha trajetória.

Aos amigos do IC e da Unicamp, em especial Fábio, Claúdio, Sheila, Viviane, Leonel, Celmar, Evandro, Luciano, Eric, Denise, pela amizade e companheirismo, e por estarem sempre lá para uma boa conversa.

Aos amigos do LAS, Martim, Felipe, Cleymone, Arthur, Edmar, Eduardo, Fernando, Ruppert, Guilherme, André, Iuri, João, Danilo, Bruno, Helen, por nossas conversas de laboratórios, seus conselhos e suas contribuição, e pela amizade estabelecida.

Aos meus amigos de Curitiba, Marcelo, Ezequiel e Julio, que mesmo após tanto tempo, continuam grandes amigos com quem posso contar.

E, finalmente, a todos os colegas, professores e funcionários do Instituto de Computação.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Desafio

A interligação entre organizações atinge proporções globais graças a Internet. A facilidade e agilidade no intercâmbio de informações melhoraram significativamente a eficiência e a competitividade de comércios, indústrias e corporações, afetando positivamente inclusive áreas como ensino e governo. Termos como *E-Business*, *E-Commerce* e *E-Learning* são forjados na literatura internacional e se tornam presentes no cotidiano, dando forma a uma "Sociedade da Informação"[32].

Entretanto, conjuntamente com as vantagens provenientes do uso cada vez mais difundido da Internet, emergiu uma crescente preocupação com a segurança de informações. O acesso por pessoas não autorizadas a informações sigilosas pode causar enormes prejuízos a uma empresa ou usuário, o que tem direcionado diversos esforços no sentido de melhorar a segurança dos sistemas computacionais envolvidos, visando impedir, ou pelo menos limitar, o acesso não autorizado.

Para garantir a segurança de sistemas computacionais, esses esforços têm sido direcionados a três focos principais de ação: segurança de redes, segurança de aplicações e segurança de sistemas operacionais.

A segurança de redes constitui a primeira barreira para ataques remotos, e visa restringir tanto o acesso às informações trafegando pela rede quanto a subversão de algum serviço que dê acesso ao computador em si. No primeiro caso, um atacante poderia obter as informações simplesmente acessando os pacotes que trafegam pela rede, e, no segundo, obter as informações diretamente de uma máquina comprometida. O uso de *firewalls*, filtros de pacotes, detectores de intrusão e a criptografia são alguns dos mecanismos utilizados com a finalidade de limitar o acesso externo [17].

Por sua vez, a segurança de aplicações constitui a segunda linha de defesa para ataques remotos. A grande maioria dos ataques que resultam em comprometimento de um

computador e, por vezes, de uma rede inteira, utiliza-se de falhas nas aplicações servidoras ou clientes de serviços de rede, permitindo ao atacante obter acesso às informações diretamente. Para minimizar as falhas presentes nas aplicações, existem técnicas de programação segura [45] e mecanismos que visam detectar tentativas de subversão de uma aplicação (por exemplo, o *StackGuard* [10]). A primeira consiste num conjunto de *guidelines* que, se seguidas, resultam num código menos propenso a falhas. O segundo consiste em plantar marcadores no meio do código, na tentativa de identificar ataques como *buffer overflow* ou *formatted string attacks*.

O terceiro foco, segurança de sistemas operacionais, foi negligenciado nos primórdios dos sistemas operacionais, e muitos sistemas operacionais modernos apresentam problemas de segurança já estudados, mas cujas soluções não foram implementadas para manter a compatibilidade com sistemas mais antigos (nestes, incluem-se o Linux e o Windows XP) [29]. A idéia, neste contexto de segurança, é restringir cada aplicação somente às informações de que necessita, e impedir qualquer acesso fora deste escopo. Com isso, mesmo que um atacante conseguisse passar pela segurança de rede e comprometer uma aplicação, ainda assim ele teria acesso muito restrito, impedindo-o de acessar informações não autorizadas.

1.2 Motivação

Muitos passos já foram dados no sentido de garantir a segurança de sistemas operacionais, porém a absorção dos sistemas operacionais seguros por usuários e organizações tem sido lenta, por vários motivos [23], entre os quais se destaca a complexidade de criação e gerenciamento de políticas de segurança para sistemas operacionais. Existem vários modelos de políticas de segurança, cada qual visando uma determinada característica de segurança (ou conjunto de características), como confidencialidade, integridade e não interferência [6]. Derivadas desses modelos, existem várias especificações de políticas de segurança para sistemas operacionais específicos, o que torna inviável gerenciar uma rede que depende de mais de uma política ou sistema operacional de forma segura.

Mesmo o tratamento de uma única política de segurança para um determinado sistema operacional é complexo, dado o nível de detalhamento necessário para configurar corretamente a segurança de determinados recursos do sistema, e a complexidade em gerenciar múltiplos domínios e perfis de usuários do sistema. A título de exemplo, a definição da política de segurança do sistema operacional SELinux (Security-Enhanced Linux) [27] disponível para o GentooLinux, para os serviços básicos do sistema operacional mais o Apache e o SSH, possui aproximadamente 7.700 linhas (sem contar comentários e linhas em branco).

Para simplificar o processo de especificação das políticas, muitos sistemas utilizam ma-

cros para os conjuntos de definições mais comuns (por exemplo, para especificar o domínio básico de uma aplicação e o acesso ao diretório temporário). Adicionalmente, para tentar garantir a corretude das políticas, existem verificadores de políticas baseados em regras, cuja finalidade é identificar e alertar o administrador das políticas sobre eventuais falhas na especificação, como é o caso da ferramenta Gokyo [21].

Entretanto, tanto as políticas de segurança quanto as restrições de verificação são definidas utilizando arquivos de texto, onde cada regra da política compreende uma ou mais linhas de definição. Apesar de permitir maior flexibilidade na especificação de políticas de segurança, essa representação dificulta o seu gerenciamento, podendo resultar em uma política incompleta, ambígua, ou mesmo conflitante. Mesmo o uso de verificadores de políticas não impede uma definição incorreta, uma vez que visam corrigir os erros mais comuns, restringindo-se à correção sintática da definição e não cobrindo características semânticas inerentes ao ambiente de uso da política.

1.3 Proposta

Considerando que um dos problemas que tem dificultado uma adoção mais ampla de sistemas operacionais mais seguros está relacionado à dificuldade de configurar corretamente as políticas de segurança desses sistemas, dada a complexidade de especificar milhares de regras descritas em formato textual e garantir que elas sejam condizentes com as finalidades de segurança a que se propõem, o objetivo desta dissertação é estudar as características inerentes aos sistemas operacionais seguros e políticas de segurança e projetar uma ferramenta que auxilie no processo de especificação.

Uma grande limitação do modelo textual de especificação é a dificuldade de compreender o impacto de uma regra em relação às demais regras no comportamento do sistema operacional, o que pode ocasionar na inserção não intencional de brechas de segurança. Adicionalmente, dada a quantidade de regras em uma política básica, a sua análise em busca de falhas conceituais ou de especificação fica relegada a verificadores automáticos, também limitados a um conjunto de regras e restrições definidas em formato textual. Portanto, decidiu-se pelo projeto de uma ferramenta para visualização gráfica de políticas de segurança, cuja finalidade é melhorar a compreensão, análise e validação das estruturas das regras, auxiliando na configuração correta do comportamento esperado do sistema operacional em relação à segurança, e que permitisse identificar problemas existentes na política especificada.

A grande variedade de sistemas operacionais seguros e modelos de política de segurança levaram à decisão de utilizar o sistema operacional SELinux e sua respectiva política de segurança como base de desenvolvimento da ferramenta, tendo em vista a flexibilidade na especificação de políticas diferentes e o uso de um modelo misto para definição de sua

política. Adicionalmente, sua compatibilidade com as aplicações já existentes do sistema operacional Linux e seus derivados permite uma migração mais fácil de usuários desses sistemas para o SELinux. Entretanto, espera-se beneficiar não só o SELinux, mas também outros sistemas operacionais seguros, projetando a ferramenta de forma a permitir futuras expansões a outros modelos e representações, ampliando seu potencial de uso.

Além do desenvolvimento do projeto, foi realizada a implementação parcial da ferramenta, consistindo de uma implementação parcial do projeto, tendo por base uma das representações criadas para apresentar a política, permitindo a leitura dos arquivos de políticas e a interação do usuário com a estrutura visual utilizada para representar a política.

1.4 Organização

Esta dissertação está estruturada em 7 capítulos, organizados da seguinte forma:

Capítulo 1: Introdução - apresenta o contexto geral deste trabalho.

Capítulo 2: Sistemas operacionais e segurança - apresenta os conceitos gerais de segurança em sistemas operacionais, detalhando arquiteturas e mecanismos que visam robustecer sua segurança.

Capítulo 3: Políticas de Segurança - apresenta os principais modelos de política de segurança e suas classificações, tratando de aspectos gerais a segurança computacional.

Capítulo 4: SELinux: Sistema e Políticas de Segurança - apresenta a estrutura de segurança do SELinux, fazendo um paralelo com os pontos apresentados nos dois capítulos anteriores.

Capítulo 5: Análise de Requisitos - apresenta a etapa de análise de requisitos realizada para identificar as características necessárias ao projeto.

Capítulo 6: PolicyViewer - apresenta o projeto da ferramenta de visualização de segurança e os detalhes da implementação parcial da ferramenta para validar os objetivos da dissertação.

Capítulo 7: Conclusão - apresenta as principais conclusões e benefícios provenientes deste trabalho, e descreve trabalhos futuros que podem ser realizados sobre o projeto.

Capítulo 2

Sistemas Operacionais e Segurança

Os sistemas operacionais foram criados com dois objetivos básicos: criar uma camada de abstração entre o *hardware* e as aplicações, e gerenciar os recursos do sistema de forma eficiente [44]. Como os computadores encontravam-se isolados entre si e poucas pessoas tinham acesso a eles, as preocupações de segurança eram mínimas, e geralmente eram negligenciadas para permitir um aproveitamento maior do desempenho limitado dos primeiros computadores.

Com o passar do tempo tanto os computadores quanto os sistemas operacionais foram se aprimorando e disseminando. Organizações começaram a interligar seus computadores em redes e os sistemas operacionais começaram a permitir a execução de processos em paralelo, inclusive de usuários diferentes. Finalmente, com o surgimento e crescimento da Internet, as redes computacionais de organizações foram se unindo, permitindo um intercâmbio de informações em escala global.

Infelizmente, os sistemas operacionais em uso atualmente baseiam muito da sua estrutura nos primeiros sistemas operacionais, o que torna a segurança provida por eles insuficiente para a Internet. Os computadores, antes isolados e acessíveis a poucos usuários, agora podem ser acessados dos mais diferentes lugares do mundo, expondo-os a inúmeros atacantes potenciais.

Neste capítulo, a questão das segurança em sistemas operacionais será apresentada. Em primeiro lugar, serão apresentadas as falhas presentes nos sistemas operacionais mais usados. Em seguida, serão analisados dois dos principais aspectos de segurança relacionados a esta dissertação: Controle de Acesso e Arquitetura de Sistemas Operacionais. Finalmente, serão discutidos os aspectos que dificultam a adoção de sistemas operacionais seguros.

2.1 Sistemas operacionais mais usados e suas falhas

Entre os sistemas operacionais mais difundidos atualmente enquadram-se o Unix e seus derivados (Linux, BSD e suas várias distribuições), o Windows (em especial o NT, XP e o 2000) e o MacOS. Derivados de sistemas operacionais de terceira geração [44], esses sistemas apresentam muitas das características de segurança de seus predecessores[1], nos quais a segurança reside nos direitos de acesso de um usuário (ou grupo de usuários) aos recursos disponíveis (arquivos, periféricos, memória, entre outros).

Para entender a necessidade de modificar os mecanismos de segurança existentes nesses sistemas operacionais é preciso inicialmente entender quais as limitações desses mecanismos.

A primeira característica a ser apontada é a maneira como é tratado o controle de acesso. Como apresentado anteriormente, os direitos de acesso são baseados no usuário que está operando o computador, ou melhor, na identidade desse usuário. Uma vez verificada a identidade de um usuário, todas as aplicações inicializadas por ele executarão em nome dessa identidade. Em função disso, todas as decisões do sistema operacional, como permitir ou não à aplicação acessar algum arquivo (ou recurso do sistema), serão tomadas analisando os direitos de acesso concedidos ao usuário sobre esse arquivo. Esse conceito de controle de acesso é denominado *IBAC* (*Identity-Based Access Control* - controle de acesso baseado em identidade)[6].

Apesar de eficiente para computadores isolados, o *IBAC* representa sérios riscos em ambientes interligados em rede. Como apresentado em [45], as aplicações estão ficando cada vez mais complexas e, conseqüentemente, mais sujeitas a vulnerabilidades em sua programação. A questão é que existem várias aplicações que são utilizadas para comunicação em rede e possíveis vulnerabilidades nelas podem ser utilizadas para obter acesso ao computador, dando ao atacante os mesmos direitos dados ao usuário cuja identidade está associada à aplicação. Por exemplo, considere o caso de um usuário executando um navegador de internet (*browser*). Eventualmente, ele pode entrar em um *site* que contenha um código malicioso preparado para explorar vulnerabilidades do navegador e tomar o controle de sua execução, expondo assim o domínio do usuário ao atacante.

Outra característica desses sistemas operacionais que influencia em sua segurança está relacionada à existência de um usuário especial, denominado administrador ou super-usuário, cuja função é gerenciar os arquivos e recursos do sistema operacional que os demais usuários não podem acessar ou modificar diretamente. Nessa classe, incluem-se as configurações do sistema operacional, o controle da instalação de aplicações tanto clientes

[1] O Windows é uma exceção, visto que derivou do sistema operacional DOS, o qual não possuía nenhum mecanismo para garantir a segurança dos recursos do sistema. Posteriormente, o Windows NT foi desenvolvido incluindo mecanismos de segurança e um sistema de arquivos que permitia o controle de acesso — o NTFS.

como servidoras, entre outros. Dessa forma, é retirada dos usuários normais a responsabilidade sobre o funcionamento normal do sistema operacional, minimizando o impacto causado por erros desses usuários. Se alguma aplicação executando sob a identidade de um usuário normal necessitar de acesso a algum recurso privilegiado do sistema operacional, o mesmo deverá ser feito através de uma interface bem definida e controlada pelo sistema operacional, denominada *API* (*Application Programming Interface*), de tal forma permitindo ao sistema operacional mediar o acesso e controlar os direitos concedidos ao usuário. O risco associado a esta característica se deve ao fato de que, assim como os usuários normais, o super-usuário pode executar aplicações, que ficam associadas a sua identidade, dando a um atacante que subverta essas aplicações controle sobre todo o sistema operacional.

A última característica a ser analisada está relacionada à forma como são implementados os sistemas operacionais. As instruções disponíveis em qualquer computador podem ser agrupadas em instruções privilegiadas e instruções não privilegiadas, e dois modos básicos de execução: protegido ou normal (alguns computadores apresentam mais modos, dividindo as instruções privilegiadas em níveis). Em modo protegido, qualquer instrução pode ser executada, enquanto em modo normal, apenas as instruções não privilegiadas são permitidas. Dessa forma, como as instruções privilegiadas incluem as instruções de controle dos recursos do computador (memória, discos e periféricos), o sistema operacional pode utilizar esta estrutura para gerir o computador. Executando em modo protegido, e permitindo às aplicações executarem somente em modo normal, ele impede que as aplicações modifiquem o comportamento do sistema. Como visto anteriormente, a *API* fica responsável por mediar o acesso das aplicações a essas instruções privilegiadas.

Infelizmente, por questões de desempenho, os sistemas operacionais incluem vários módulos de código que são executados em modo protegido, responsáveis por gerenciar os mais diversos recursos, como sistema de arquivos, de rede, *drivers* de periféricos. Esse fator adiciona complexidade ao código do sistema operacional, de tal forma que, assim como no caso das aplicações, torna-se impossível garantir que não haja vulnerabilidades que possam ser exploradas. Só que, neste caso, o atacante obterá acesso direto às instruções privilegiadas, inutilizando qualquer tentativa de barrar o ataque a partir de aplicações seguras.

Nas próximas seções, serão analisadas em mais detalhes as questões do controle de acesso e da arquitetura dos sistemas operacionais, já levando em conta as características necessárias a um sistema operacional mais seguro.

2.2 Controle de Acesso

Para entender o que é e como funcionam os mecanismos de controle de acesso de um sistema operacional, faz-se necessário classificar os elementos do sistema em três grupos. O conjunto de *sujeitos* do sistema operacional englobam qualquer elemento que pode realizar ações sobre objetos, e é composto por usuários, processos e o próprio sistema operacional. Por sua vez, o conjunto de *objetos* do sistema operacional englobam os elementos sobre os quais podem ser realizadas ações, e inclui, entre outros, arquivos, endereçamentos de memória, recursos do sistema e processos. É importante destacar que os processos pertencem aos dois conjuntos, uma vez que outro processo ou o sistema operacional pode realizar ações sobre ele (por exemplo, mandar um sinal de término de execução forçada). O terceiro conjunto, o de *ações*, compreende a lista de ações que podem ser realizadas sobre cada um dos objetos do sistema operacional. Por exemplo, ler, escrever e apagar são ações que podem ser realizadas sobre um arquivo, enquanto abrir conexão, enviar mensagem e encerrar conexão são ações que podem ser realizadas em um canal de comunicação em rede.

Por uma questão de clareza, no restante da dissertação será utilizado o termo *processo* para definir uma aplicação em execução no sistema operacional, e o termo *aplicação* para definir o arquivo em disco que contém o código executável.

Portanto, determinar se um sujeito tem direito de realizar uma ação sobre um objeto e permitir ou não a realização desta é o objetivo dos mecanismos de controle de acesso do sistema operacional. Partindo dessa definição, surgem dois novos conceitos: permissão e domínio. Uma *permissão* é o direito que um sujeito tem de realizar uma ação sobre um objeto, e será identificada ao longo desta dissertação pelo mesmo nome da ação. Um *domínio* é o conjunto de todas as permissões sobre objetos do sistema operacional associadas a um sujeito.

Tratar o controle de acesso em função de processos, e não só em função da identidade dos usuários, permite implementar no sistema operacional o princípio de privilégio mínimo (*least privilege principle*). Esse princípio de segurança define que a um sujeito sejam dados apenas os privilégios necessários à realização de sua atividade. Dessa forma, em caso de comprometimento de um processo, o conjunto de objetos que sofrerá quebra de segurança será o mínimo possível. Usando o exemplo do navegador apresentado anteriormente, segundo o princípio de privilégio mínimo ele deveria ter apenas acesso de leitura às suas configurações e acesso de leitura e escrita no diretório de arquivos temporários, (desconsiderando-se as complicações referentes ao uso de *plug-ins*). Assim, mesmo que o navegador seja comprometido (e seu respectivo domínio), o domínio do usuário não seria afetado.

A maioria dos sistemas operacionais garante que os usuários e processos só realizem

ações utilizando a arquitetura em dois modos de execução explicada anteriormente (modos normal e protegido), e verificando as permissões através das chamadas da *API* disponibilizada às aplicações.

Os mecanismos de controle de acesso mais utilizados são baseados em quatro modelos: Matriz de Controle de Acesso, Listas de Controle de Acesso (ou *ACL*, sigla de *Access Control List*), *Capabilities* e sistema de cadeados-chaves (*Lock and Key System*).

2.2.1 Modelo de Matriz de Controle de Acesso

A forma mais precisa para definir as permissões do sistema operacional é utilizar o modelo de matriz de controle de acesso proposta em 1971 por Lampson [25], que descreve as permissões dos processos sobre arquivos em uma matriz.

Neste modelo, cada linha representa um sujeito do sistema operacional, e cada coluna representa um objeto. As células contém o conjunto de permissões que o sujeito tem sobre o objeto. Por exemplo, a Tabela 2.1 apresenta as regras de controle de acesso de um sistema operacional fictício. O processo 1 pode ler dos arquivos 1 e 2, e escrever no primeiro, porém não tem nenhuma permissão de acesso ao arquivo 3. Adicionalmente, ele pode se comunicar com o processo 2 enviando mensagens. O processo 2, por sua vez, pode ler dos arquivos 2 e 3, escrever nos arquivos 1 e 3, e receber mensagens do processo 1.

	Arquivo 1	Arquivo 2	Arquivo 3	Processo 1	Processo 2
Processo 1	ler, escrever, possuir	ler	—	ler, escrever executar, possuir	escrever
Processo 2	escrever	ler, possuir	ler, escrever	ler	ler, escrever, executar, possuir

Tabela 2.1: Exemplo de matriz de controle de acesso. O sistema possui 2 processos e 3 arquivos, e o conjunto de permissões disponíveis é {ler, escrever, executar, possuir}

A interpretação do significado dessas permissões pode variar de sistema para sistema. Ler e escrever arquivos é usualmente claro, porém o significado de "ler de um processo" pode variar. No exemplo acima, foi assumido que essa permissão concede ao processo 2 o recebimento de mensagens do processo 1, mas poderia também representar comunicação através do uso de uma área de memória comum aos dois processos.

A permissão "possuir" é um direito que se distingue dos demais. Na maioria dos sistemas, o criador de um objeto tem o privilégio especial de poder adicionar e remover permissões sobre o objeto para outros sujeitos. No exemplo da Tabela 2.1, cada processo

possui a si próprio e ao arquivo de número correspondente. Portanto, o processo 2 poderia conceder ao processo 1 a permissão de escrever no arquivo 2, ou mesmo revogar o acesso de leitura a este arquivo. Esse tipo de controle de acesso é denominado Controle de Acesso Discricionário ou *DAC* (*Discretionary Access Control* [13]), uma vez que as decisões referentes as permissões sobre objetos ficam à discrição do usuário que o criou.

Outros sistemas possuem ainda políticas que determinam regras de controle de acesso. Assim, mesmo que um usuário crie um objeto e deseje liberar o acesso a este para outro usuário, o sistema só permitirá o acesso se a política não proibí-lo. Esse tipo de controle de acesso é denominado Controle de Acesso Mandatório ou *MAC* (*Mandatory Access Control*), e é utilizado paralelamente com o DAC na maioria dos sistemas operacionais seguros.

Apesar de o exemplo ter utilizado como objetos somente arquivos e processos como objetos, os objetos definidos em um sistema operacional incluem também todos os recursos disponíveis aos processos através da API (*Application Programming Interface*) do sistema operacional.

Embora precisa, a Matriz de Controle de Acesso é útil apenas como modelo teórico, pois sua implementação direta apresenta inúmeros problemas. Num sistema típico, o número de sujeitos e objetos é suficientemente grande para que a matriz utilize muito espaço para seu armazenamento, e muitas das entradas da tabela serão brancas ou iguais, indicando inexistência de acesso e definições de acessos padrões aos objetos, respectivamente. Além da questão do espaço necessário, a criação e remoção de sujeitos e objetos requerem gerenciamento cuidadoso da matriz, o que torna a implementação do mecanismo muito complexa. Por isso, foram desenvolvidas otimizações para permitir uma implementação conveniente, e em alguns casos mais simples, do modelo.

2.2.2 *Access Control Lists*

Uma variação direta da matriz de controle de acesso é armazenar cada coluna da tabela com o objeto que ela representa, e é denominado Lista de Controle de Acessos (*ACL - Access Control List*). Assim, cada objeto está associado a um conjunto de pares, cada par contendo um sujeito e um conjunto de permissões. Por exemplo, a *ACL* apresentada na Tabela 2.2 representa o mesmo conjunto de permissões do sistema fictício apresentado na Tabela 2.1.

Um problema associado ao uso de *ACL*s em sistemas com muitos sujeitos é que a lista pode ser muito grande, fazendo-se necessário utilizar abreviações na lista. Definições padrões para sujeitos não presentes na lista, grupos de usuário e caracteres "coringas" são algumas das formas utilizadas para abreviar a lista. O UNIX [38], por exemplo, divide os conjuntos de usuários do sistema em três classes: o proprietário do arquivo,

Objeto	*ACL*
Arquivo 1	(Processo 1, {ler, escrever, possuir}), (Processo 2, {escrever})
Arquivo 2	(Processo 1, {ler}), (Processo 2, {ler, possuir})
Arquivo 3	(Processo 2, {ler, escrever})
Processo 1	(Processo 2, {ler})
Processo 2	(Processo 1, {escrever})

Tabela 2.2: Exemplo de *ACL*. Esta *ACL* descreve as mesmas permissões apresentadas na Tabela 2.1

um grupo associado ao arquivo, e os demais usuários, sendo que cada classe tem um conjunto separado de permissões. Entretanto, essa divisão em apenas três classes resulta em perda de granularidade no controle de acesso, pois certos arranjos de permissões não são possíveis nesse modelo.

2.2.3 *Capabilities*

A segunda forma de simplificar o modelo de matriz de controle de acesso é associar ao sujeito as linhas da matriz, e este modelo é conhecido como *Capabilities* (Capacidades ou Habilidades). Conceitualmente, a cada sujeito está associado um conjunto de pares, cada par contendo um objeto e um conjunto de permissões e correspondendo a uma *capability* (Tabela 2.3). Por questão de uniformidade com a nomenclatura convencional, será utilizado o termo em inglês para definir este modelo.

Sujeito	*Capabilities*
Processo 1	(Arquivo 1, {ler, escrever, possuir}) (Arquivo 2, {ler}) (Processo 1, {ler, escrever, executar, possuir}) (Processo 2, {escrever})
Processo 2	(Arquivo 1, {escrever}) (Arquivo 2, {ler, possuir}) (Arquivo 3, {ler, escrever}) (Processo 1, {ler}) (Processo 2, {ler, escrever, executar, possuir})

Tabela 2.3: Exemplo de *Capabilities*. Esta tabela descreve as mesmas permissões apresentadas na Tabela 2.1

O sistema operacional EROS é um exemplo de sistema operacional com controle de acesso baseado em *Capabilities* [41]. Quando um processo apresenta uma *capability*, o sistema operacional a examina para determinar se o processo tem a permissão desejada

para acessar um determinado objeto. Isto reflete como funciona *capabilities* para gerenciamento de memória: a localização de um objeto na memória é encapsulada em uma *capability*. Sem esta, o processo não pode nomear o objeto de forma a obter o acesso desejado.

A arquitetura de *capabilities* é mais interessante em termos de implementação que a de *ACL*. Em sistemas que implementam *ACL*, tanto a lista de controle de acesso quanto a identidade do processo estão sob o controle do sistema operacional, de tal forma que o usuário só pode manipulá-las por intermédio da *API*. Já em sistemas que implementam *capabilities* o processo precisa identificar uma *capability* para poder utilizá-la, tendo portanto controle sobre ela. Se o processo puder forjar uma *capability* e utilizá-la, o controle de acesso irá falhar. Existem três alternativas conhecidas para proteger as listas de *capabilities*: marcas, memória protegida e criptografia.

A proteção por marcas possui um conjunto de *bits* associado a cada endereçamento de memória, sendo que a marca tem dois estados possíveis: somente leitura ou leitura e escrita. Apenas processos privilegiados podem modificar o estado da marca. Um exemplo de arquitetura com marcas é o sistema B5700 [6]. A proteção por memória protegida consiste em utilizar marcas associadas a páginas ou segmentos da memória. Todas as *capabilities* são armazenadas em uma página cujo processo pode ler, mas não alterar. Por não necessitar de um *hardware* específico, esta alternativa é mais comum. O sistema CAP [33] é um exemplo de sistema que utiliza essa abordagem. A terceira alternativa utiliza criptografia para gerar uma assinatura para as *capabilities*, identificando tentativas de modificação pela verificação das assinaturas associadas. O sistema Amoeba [31] utiliza essa alternativa para o controle de acesso de objetos em um sistema distribuído.

2.2.4 Cadeados e Chaves

O último mecanismo de controle de acesso a ser analisado é conhecido como técnica dos cadeados e chaves (*Locks and Keys Technique*), e combina características de *ACL* e *capabilities*. Neste mecanismo, uma parte de informação (o cadeado ou trava) é associada ao objeto, e uma segunda parte da informação (a chave) é associada ao sujeito autorizado a acessar o objeto. Quando o sujeito tenta acessar um objeto, o conjunto de chaves do sujeito é analisado para encontrar uma chave que corresponda exatamente ao cadeado do objeto, e só então permitindo o acesso. Uma sugestão de implementação de cadeados e chaves com criptografia é apresentada em [19].

2.3 Arquitetura de Sistemas Operacionais Seguros

Como visto anteriormente, os sistemas operacionais atuais derivam sua arquitetura de sistemas operacionais mais antigos, e os mecanismos de segurança foram sendo incorporados a uma estrutura que não previa os requisitos de segurança do mundo atual. O problema por trás dessa decisão de projeto é similar ao de desenvolver um sistema de alto-desempenho a partir de um sistema com desempenho ruim. Se o desempenho ruim é atribuído a funções específicas, essas funções podem ser reprojetadas. Entretanto, a estrutura, projeto e estilo do sistema estão provavelmente na raiz do problema de desempenho, e corrigir esses elementos do sistema é um trabalho bem mais complexo. Seria melhor recomeçar o projeto do zero, considerando desde o início desempenho como objetivo primário.

Por esse motivo, pesquisadores apontam para as falhas de segurança existentes nos sistemas operacionais em uso, uma vez que as suposições de segurança em que se baseiam são insuficientes para garantir a segurança das aplicações e informações [29]. Várias arquiteturas foram propostas, com as arquiteturas *SawMill* [18], Birlix [20], *Trusted Computing Base* (*TCB*) do *Trusted Computing Group* [24]. Este último, em especial, propõe a inclusão de *hardware* para gerenciar as operações de segurança, impedindo assim que os mecanismos de segurança possam ser corrompidos. Várias arquiteturas tornaram-se sistemas operacionais funcionais, e incluem-se nesse grupo, além dos sistemas já citados: SPIN Operating System [4], Flask [43], Perseus [36] e SELinux [27, 28]. Nesta seção, serão abordadas algumas das características que conferem segurança a esses sistemas operacionais.

Na maioria dos sistemas operacionais, normalmente o controle de segurança está baseado nas operações de acesso aos dados, como leitura e escrita. Outros, como no caso do Birlix [20], por ser baseado no modelo de programação orientada a objetos, controla as invocações de métodos dos objetos do sistema. Ou ainda, como no caso do Mach, utilizam a interceptação de chamadas entre processos (*IPC - Inter-process call*) para controlar a segurança do sistema. Em qualquer um dos casos, a granularidade do controle das operações deve ser analisada, uma vez que influencia diretamente na segurança do sistema operacional. Por exemplo, o controle de acesso a arquivos no Unix [38] é feito em função de três tipos de permissões: leitura, escrita e execução. Uma aplicação de *log*, mesmo precisando apenas escrever ao final do arquivo de *log*, poderá modificar qualquer porção desse arquivo, o que pode ser utilizado por um atacante que obteve acesso ao mesmo domínio para apagar os rastros da invasão, ocultando sua ação. Entretanto, o excesso de granularidade pode causar impacto no desempenho do sistema operacional, já que cada operação controlada deverá ser analisada para verificar se o processo envolvido tem permissão para realizá-lo.

Um conceito importante, presente em todos os sistemas operacionais seguros, é o Mo-

nitor de Referência (*Reference Monitor*) [13]. Um monitor de referência é um mecanismo de validação que media todos os acessos a objetos realizados por sujeitos do sistema operacional, e é responsável por impor as políticas de segurança sobre todas as ações realizadas. Nesse sentido, o monitor de referência deve ser sempre invocado (evitando, assim, tentativas de contornar os mecanismos de segurança), ser a prova de falhas e deve ser pequeno o suficiente para ser submetido a uma análise de corretude e testes, para que sua eficácia seja comprovada.

A questão da análise de corretude e realização de testes não se restringe somente ao monitor de referência. Vários sistemas operacionais seguros têm sido desenvolvidos utilizando uma estrutura de *microkernel*, em que o núcleo do sistema operacional é responsável apenas pelos aspectos essenciais do computador: gerenciamento de processos e memória, e intermediação de acesso ao *hardware*. Todas as demais funcionalidade, como sistemas de arquivos e controle de periféricos, são tratadas por aplicações executando no mesmo espaço que o dos usuários. Dessa forma, obtém-se um núcleo menor e verificável utilizando técnicas como análise de corretude e testes exaustivos, o que confere a estes maior confiabilidade do que um núcleo como o do Linux. Neste último, a grande quantidade de subsistemas diretamente ligados ao núcleo tornam inviável uma análise de corretude, e podem conter vulnerabilidades que comprometam toda a segurança do sistema.

O uso de *microkernel* também apresenta outras vantagens além da garantia de funcionamento correto do núcleo. Como nem todos os subsistemas do sistema operacional estão ligados diretamente ao núcleo, eles podem ser analisados individualmente, aumentando a confiança geral no comportamento do sistema. Mesmo o monitor de referência não precisa estar associado ao núcleo, desde que o sistema operacional imponha a execução dele a cada operação realizada. Assim, mesmo que algum subsistema seja comprometido, todos os demais estarão seguros.

2.4 Dificuldades na Adoção de Sistemas Operacionais Seguros

Apesar de já existir uma ampla variedade de sistemas operacionais, utilizando os mais variados sistemas de controle de acesso e modelos de políticas de segurança, a adoção de sistemas operacionais seguros ainda é incipiente na maioria dos ambientes computacionais de organizações e em computadores pessoais.

Um dos principais fatores está relacionado à migração de aplicações de um sistema operacional para outro, que impõe uma certa "inércia" à substituição dos sistemas operacionais [23]. As aplicações, de um modo geral, são desenvolvidas para utilizar da melhor maneira possível as funcionalidades e estruturas providas pelo sistema operacional e, por-

tanto, estão fortemente ligadas ao sistema operacional para o qual foram desenvolvidas. A sua utilização direta em outro sistema operacional, portanto, não será possível se as estruturas providas forem incompatíveis, gerando a necessidade de adquirir novos programas para suprir as necessidades da organização. O alto custo financeiro e o longo tempo necessário para desenvolvimento e treinamento das novas aplicações tornam inviável a migração de sistemas operacionais.

Sendo a substituição do sistema operacional muito custosa, a solução seria adaptar os sistemas operacionais existentes para se adequarem a requisitos de segurança. Entretanto, como apresentado anteriormente, essa solução pode implicar reestruturação de boa parte do sistema operacional ou, em caso ainda pior, não ser possível comprovar que os mecanismos de segurança são realmente eficientes. Além do núcleo do sistema operacional em si, outros sistemas auxiliares também poderão sofrer remodelagem para se adaptar aos requisitos de segurança. Por exemplo, o sistema de arquivos necessitará armazenar informações adicionais sobre as características de segurança associadas aos arquivos, para permitir um controle mais efetivo. Atualmente, uma das grandes promessas nesse sentido é o SELinux (*Security-Enhanced Linux*) [27, 28], que será visto em mais detalhes no Capítulo 4.

Outro fator que pesa na adoção de sistemas operacionais seguros é a complexidade envolvida no gerenciamento de políticas de segurança. Como apontado por Baker [1], as políticas, em geral, são de difícil compreensão, pouco escaláveis e limitadas em termos de usuários, função, operações e classes. Esse ponto será melhor abordado no Capítulo 3.

2.5 Conclusão

Neste capítulo, foram apresentadas características inerentes a sistemas operacionais seguros. Foram apresentadas as características presentes nos sistemas operacionais mais difundidos, analisando suas falhas no atual ambiente computacional em que estão inseridos. O conhecimento das suas limitações é essencial para entender a necessidade de sistemas operacionais mais seguros e a importância de incentivar a sua adoção.

O estudo dos mecanismos de controle de acesso e suas características é importante para entender como o sistema operacional gerencia as ações realizadas por processos aos arquivos e recursos do sistema operacional, e permite perceber as vantagens e desvantagens de utilizar uma solução ao invés das outras. Requisitos distintos de segurança podem necessitar de soluções diferentes, e a compreensão de como o controle de acesso é realizado permite escolher a solução mais adequada.

A análise de questão de arquitetura dos sistemas operacionais, das estruturas utilizadas para garantir a segurança das aplicações e dos objetos do usuário, permite o estudo dos variados sistemas operacionais seguros existentes, auxiliando na escolha daqueles mais

seguros com relação a um determinado aspecto de segurança.

Entretanto, a adoção desses sistemas operacionais seguros ainda esbarram em várias questões técnicas, e mesmo econômicas, que são apresentadas ao final do capítulo, e que motivam o desenvolvimento de ferramentas que auxiliem uma migração mais fácil para paradigmas mais seguros.

Capítulo 3

Políticas de Segurança

O desenvolvimento de sistemas operacionais seguros criou a necessidade de se desenvolver políticas de segurança para especificar os critérios de segurança desejados. As *ACL's* e *Capabilities* são precisas ao especificar as permissões do domínio de cada usuário e processo para o sistema operacional, porém faltam-lhes legibilidade para o administrador de segurança.

No intuito de simplificar a especificação de políticas, torná-las mais claras aos usuários e assim evitar eventuais erros na especificação que comprometam a segurança, foram desenvolvidas abstrações para representar a política de segurança. Neste capítulo, são analisados os principais modelos de políticas de segurança, baseados nos conceitos básicos de segurança computacional.

3.1 Conceitos Básicos

Segurança computacional pode ser definida de várias formas, dependendo do contexto em que é tratada. Pode variar em função da estrutura computacional, das pessoas envolvidas, ou mesmo das regras de uma organização em particular, porém, de um modo geral, os objetivos de segurança podem ser resumidos a três conceitos básicos: confidencialidade (*confidentiality*), integridade (*integrity*) e disponibilidade (*avaliability*) [6].

Confidencialidade consiste, basicamente, em impedir o acesso não autorizado a informações ou recursos do sistema. Instituições militares e industriais são os principais exemplos para a necessidade de se manter uma informação em sigilo. O vazamento de informações sobre posicionamento de tropas pode resultar na reversão da vantagem estratégica em uma guerra, enquanto que uma indústria pode perder a vantagem competitiva se outra indústria obtiver acesso ao projeto de novos produtos.

É importante observar que confidencialidade não se restringe somente a proteger a informação contida em um arquivo do sistema operacional. Em determinados contextos,

a confidencialidade aplica-se também à existência da informação, que pode ser mais relevante do que a informação propriamente dita. Por exemplo, a descoberta da existência de um plano de ataque pode ser utilizado pelo país alvo para reforçar as defesas, mesmo que o plano seja desconhecido.

Por sua vez, *integridade* refere-se ao nível de confiança com relação a corretude de uma informação ou recurso, podendo também ser definido em função da garantia de que estes não sofreram alterações impróprias ou não autorizadas. Diferentemente da confidencialidade, garantir a integridade de um sistema é um processo muito mais complexo, pois depende de dois fatores: integridade dos dados (o conteúdo da informação) e integridade da origem (a fonte dos dados). Este último influencia na precisão e na credibilidade dada às informações armazenadas.

Ao se tratar sobre integridade em um sistema computacional, deve-se observar que as informações não são corrompidas necessariamente por acessos não autorizados. Um usuário ou processo com acesso autorizado pode alterar o conteúdo de uma base de dados de forma incorreta (acidental ou propositalmente), de tal forma que se faz necessário tratar a integridade das informações para qualquer tipo de acesso. Por exemplo, um caixa de banco possui acesso às contas dos clientes, para poder realizar as transações requeridas. Entretanto, sem mecanismos de proteção adequados, ele pode utilizar esse acesso para transferir dinheiro para uma conta pessoal, fraudando o banco.

Finalmente, *disponibilidade* refere-se à possibilidade de utilizar uma informação ou recurso quando desejado. É um importante aspecto de confiabilidade do sistema (*reliability*), já que um sistema indisponível é pelo menos tão ruim quanto sistema nenhum. O aspecto de disponibilidade relevante para segurança computacional está relacionado ao fato de que alguém pode deliberadamente provocar a negação de acesso a informações. Tentativas de bloquear o acesso, denominados ataques de negação de serviço (*deny of service attacks*) são de difícil detecção. Normalmente, a detecção é feita a partir de análises estatísticas de acesso ao serviço, podendo ocorrer falsos positivos em padrões atípicos de acesso, e ataques deliberados podem parecer padrões normais de acesso, gerando falsos negativos.

Destes três conceitos básicos, apenas os dois primeiros são de interesse ao analisar políticas de segurança para sistemas operacionais, uma vez que disponibilidade normalmente envolve o uso de sistemas redundantes, em uma arquitetura distribuída.

3.2 Políticas de Segurança

Ao analisar um sistema computacional como um autômato de estados finitos com um conjunto de funções de transição de estados, define-se um *sistema seguro* como um sistema que inicializa em um estado seguro e não pode entrar em um estado não seguro.

Entretanto, como cada organização e ambiente computacional possuem objetivos próprios de segurança, é necessário definir de forma clara e precisa quais os estados considerados seguros ou não. Portanto, tem-se que uma *política de segurança* especifica a divisão dos estados possíveis do sistema em estados autorizados (ou seguros) e estados não autorizados (ou inseguros).

Dada a complexidade de se definir a política de segurança completa de forma precisa, não existe um modelo de política adequado a todas as organizações. A grande variabilidade de requisitos tornou necessário desenvolver políticas que privilegiassem um ou mais aspectos específicos da segurança do sistema operacional. Nesse sentido, surgiu a classificação de políticas em *políticas militares* (ou *governamentais*) e *políticas comerciais*. A primeira classificação engloba políticas desenvolvidas primariamente para prover confidencialidade às informações, tendo o nome se originado da necessidade de militares e governos manterem informações em segredo. Por sua vez, a segunda engloba políticas desenvolvidas com o objetivo primário de prover integridade, fator mais importante que confidencialidade para a maioria das organizações da área comercial.

As políticas de segurança podem ser classificadas também em função do aspecto de segurança computacional para o qual foram desenvolvidas. Uma política é denominada *política de confidencialidade* se especifica somente confidencialidade das informações, e é denominada *política de integridade* se trata apenas da integridade de informação. Uma política que trate de ambos os aspectos, em qualquer proporção, é denominada *política híbrida*. Repare que uma política de confidencialidade é uma política militar, mas a afirmação inversa não é válida, uma vez que políticas militares podem tratar também de integridade.

A seguir, serão apresentados os principais modelos de segurança, analisando os aspectos mais específicos tratados por cada um.

3.2.1 Políticas de Confidencialidade

O *modelo de Bell-LaPadula* [2, 3] é o principal modelo de política de confidencialidade, e corresponde ao modelo de classificação militar americano. Ele tem influenciado o desenvolvimento de muitos outros modelos e muitas das tecnologias de segurança de computadores.

Nesse modelo, são definidos níveis de confidencialidade em uma ordenação linear, que serão associados aos sujeitos e objetos do sistema. O nível associado a um sujeito é denominado *liberação*, enquanto o nível associado a um objeto é denominado *classificação*. Esta última denominação também é usada ao se referir ao nível de sujeitos e objetos indistintamente. O objetivo da política, portanto, é prevenir acessos de leitura a objetos em uma classificação maior que a liberação do sujeito. Na Tabela 3.1 é apresentando

um exemplo com quatro níveis de confidencialidade, quatro usuários e quatro tipos de arquivos.

Classificação	Sujeitos	Objetos
Super-Secreto (SS)	João	Arquivos de Pessoal
Secreto (S)	Mariana	Arquivos de *e-mail*
Confidencial (C)	Tomás	Arquivos de *Log*
Liberados (L)	Ursula	Arquivos comuns

Tabela 3.1: Exemplo de modelo Bell-LaPadula em um sistema fictício.

Para garantir o objetivo da política, os acessos aos arquivos só são autorizados sob duas condições. A primeira condição, denominada *condição de segurança simples*, define que um sujeito pode ler um objeto se e somente se sua liberação for maior ou igual à classificação do objeto. Na Tabela 3.1, por exemplo, o usuário Tomás pode ler os arquivos de *log* e os arquivos comuns, mas não pode ler os arquivos de *e-mail* nem os de Pessoal. Entretanto, apenas essa condição não é suficiente para garantir a confidencialidade, pois caso Mariana ou João lerem os arquivos de *e-mail* e os copiassem para os arquivos de *log*, Tomás obteria acesso a esses arquivos, quebrando a regra de confidencialidade. Para impedir essa situação, uma segunda condição determina que um sujeito pode escrever em um objeto se e somente se sua liberação for menor ou igual à classificação do objeto. Esta condição é conhecida como *propriedade estrela* (**-property* ou *star property*). Assim, nem Mariana nem João poderiam copiar as informações contidas nos arquivos de *e-mail* para os arquivos de *log*, garantindo-se, dessa forma, que não há quebra de confidencialidade indiretamente.

O modelo possui ainda outras duas características. A primeira é a combinação de controle de acesso mandatório e discricionário, de tal forma que o acesso só é permitido se, além de obedecer às duas condições, o sujeito tiver permissão discricionária sobre o arquivo. A segunda consiste na inclusão de categorias na classificação de segurança, em que cada categoria descreve um tipo de informação. Assim, a cada objeto e sujeito são associadas uma ou mais categorias, distribuídas de tal forma que o sujeito só possa acessar os objetos necessários à realização de sua função (esse requisito é conhecido como princípio do "*need to know*"). Levando essas características em conta, as duas condições podem ser redefinidas da seguinte forma:

- **Condição de Segurança Simples:** Um sujeito S pode ler um objeto O se e somente se as seguintes três condições forem válidas: a liberação de S é maior ou igual à classificação de O; o conjunto de categorias associadas a O é um subconjunto das categorias associadas a S; e S tem permissão de leitura discricionária sobre O.

- **Propriedade estrela:** Um sujeito S pode escrever em um objeto O se e somente se as seguintes três condições forem válidas: a liberação de S é menor ou igual à classificação de O; o conjunto de categorias associadas a S é um subconjunto das categorias associadas a O; e S tem permissão de escrita discricionária sobre O.

A influência do modelo de Bell-LaPadula permeia todas as modelagens de segurança em segurança computacional. Foi o primeiro modelo matemático a capturar atributos de sistemas reais em suas regras. Tornou-se a base de vários padrões, incluindo o *Trusted Computer System Evaluation Criteria* (*TCSEC*, também conhecido como o "livro laranja"), desenvolvido pelo Departamento de Defesa dos Estados Unidos. Outros modelos de confidencialidade surgem de métodos práticos, podendo não ocorrer múltiplos níveis no mesmo sentido do Bell-LaPadula. Por também incluírem questões de integridade, são tratados na Seção 3.2.3, de políticas híbridas.

3.2.2 Políticas de integridade

Existem três trabalhos principais estudando a natureza de políticas de integridade. Em 1977, Biba [5] propôs três tipos de políticas, uma das quais corresponde ao dual matemático do modelo de Bell-LaPadula. Cinco anos depois, Lipner retomou o modelo de Bell-LaPadula e o combinou com o modelo de Biba para criar uma política mais adequada aos requisitos de políticas comerciais [26]. Em 1987, David Clark e David Wilson desenvolveram um modelo de integridade [9] diferente dos anteriores, utilizando transações como operações básicas e modelando sistemas comerciais mais realisticamente que os anteriores.

Antes de detalhar alguns desses modelos, é importante ressaltar o papel do fator confiança. O nível de confiança dado às informações contidas em um computador depende de quão acuradas e precisas (com relação a alguma métrica) essas informações estão. Mesmo com o uso de métricas, esse nível possui um valor subjetivo, e pode ser influenciado por outros fatores, como o usuário responsável por sua entrada ou os processos que manipularam essas informações. Em um banco, por exemplo, uma transação realizada por um atendente (menor nível de integridade) seria considerada "menos confiável" do que a realizada por um gerente (maior nível de integridade). De forma semelhante, um sistema de controle de transações bancárias, sobre o qual tenham sido realizados testes exaustivos e análise de corretude, teria maior nível de integridade que outro sistema que não passou por tais análises. Uma conseqüência dessa marcação de níveis de integridade é que, sempre que um processo realizar ações sobre alguma informação, a informação resultante do processamento terá o nível de confiança equiparado ao menor nível. Se a precisão das informações não for confiável, mesmo um programa confiável não obterá resultados corretos. De forma análoga, se um programa não é confiável, mesmo com informações confiáveis ele não obterá resultados corretos.

Todas as três políticas propostas por Biba em [5] apresentam o mesmo princípio com relação a escrita de objetos e execução de novos processos, que estão diretamente ligados ao conceito de integridade apresentado no parágrafo anterior. Um processo com nível de integridade $i(p)$ só pode escrever em objetos com nível de integridade $i(o) \leq i(p)$, e só pode inicializar um novo processo com nível de integridade $i(p_2) \leq i(p)$, no sentido de que um processo só pode gerar informações com nível de integridade igual ou inferior ao seu próprio.

As três políticas diferem com relação ao comportamento do sistema nas operações de leituras de objetos. Na *política de marcas d'água*, os processos podem baixar de nível de integridade. Toda vez que um processo lê um objeto com nível de integridade menor que o seu, o seu nível de integridade é igualado ao nível do objeto. A idéia dessa regra é que, se o processo está dependendo de dados menos confiáveis, a confiabilidade de seus resultados também é menos confiável. Dessa forma, essa política impede modificações diretas e indiretas em objetos que poderiam reduzir a suas confiabilidades, ou seja, seus níveis de integridade. Em contrapartida, essa política apresenta um problema sério: o nível de integridade dos processos somente decresce, o que significa que em algum momento eles não poderão acessar nenhum objeto de nível alto. Na *política de anel* (*ring policy*), modificações indiretas são ignoradas e o foco recai em modificações diretas, de tal forma que qualquer processo pode acessar qualquer objeto do sistema operacional. Vale ressaltar que expor todos os objetos do sistema não é uma preocupação desse modelo, uma vez que não trata da questão da confidencialidade.

A terceira política proposta por Biba é denominada *política de integridade severa* (*Strict Integrity Policy*), e corresponde a um dual matemático do modelo de Bell-LaPadula, sendo mais comumente chamado de "modelo de Biba". Essa política, assim como a de marcas d'água, previne modificações indiretas de entidades sem autorização, porém restringe o problema de rebaixamento de nível de integridade permitindo que um processo só leia objetos com nível de integridade maior ou igual ao seu. Ao acrescentar o conceito de categorias e acesso discricionário, tem-se o dual completo do modelo de Bell-LaPadula.

O modelo de integridade criado por Lipner [26] é mais adequado às necessidades comerciais de integridade, e foi baseado nos modelos de Bell-LaPadula e de Biba. Nesse modelo, o modelo de Bell-LaPadula é utilizado para definir as categorias de acesso às aplicações e arquivos do sistema em função de quatro tipos de usuários (normais, desenvolvedores de aplicações, desenvolvedores de sistema e administradores) e os níveis de segurança são utilizados para separar os arquivos de auditoria (maior nível) dos arquivos gerais do sistema (menor nível). O modelo de Lipner usa o modelo de Biba para manter isolados os dados e aplicações utilizados para desenvolvimento e os utilizados na produção da empresa. Assim, não se corre o risco de que um programa em desenvolvimento aciden-

talmente corrompa a integridade dos dados de produção. Somente quando o programa estiver completamente desenvolvido e verificado, um usuário com privilégios especiais poderá mudá-lo de categoria, desassociando o programa do desenvolvimento e associando-o a produção.

O modelo de Clark-Wilson [9] não é de grande interesse para sistemas operacionais, sendo mais aplicável em sistemas comerciais baseados em transações. Em [37], é descrita uma implementação desse modelo no sistema operacional Unix, utilizando usuários "fantasmas" para gerir as transações sobre os dados restringidos pelo modelo. Entretanto, como o próprio autor apresenta, existem problemas de segurança envolvidos no uso dessa implementação. Como o acesso aos usuários "fantasmas" é realizado através de aplicações com o bit *setuid*[1] ativo, o risco de que eles possam ser corrompidos para obter acesso a privilégios inapropriados aumenta. Paralelamente, o super-usuário pode assumir a identidade de qualquer usuário "fantasma", invalidando os requisitos de segurança do modelo.

3.2.3 Políticas Híbridas

Poucas organizações limitam seus objetivos de segurança somente em integridade ou confidencialidade. A maioria deseja ambos, em alguma proporção. Esse é um dos motivos pelos quais normalmente as políticas híbridas são mais comumente implementadas em sistemas operacionais seguros do que políticas que só tratem de confidencialidade ou integridade. Nessa seção, serão tratados três modelos de políticas: Modelo Muralha da China (*Chinese Wall Model*), *DTE* (*Domain and Type Enforcement*) e *RBAC* (*Role-Based Access Control*).

O Modelo Muralha da China [7] descreve políticas que envolvem conflito de interesse em transações comerciais, e é tão importante para essa área quanto o modelo de Bell-LaPadula é para o militar. Bolsas de ações e casas de investimento são os ambientes mais naturais desse modelo. Nesse contexto, o objetivo do modelo é prevenir situações em que um negociador, representando dois clientes com interesses conflitantes, não possa ajudar um dos clientes a ganhar lucros às custas do outro. Esse objetivo é atingido organizando as informações em conjuntos funcionais e agrupando esses conjuntos em classes de conflito. No caso de bolsa e valores, por exemplo, podem-se organizar as informações de um determinando banco A em um conjunto funcional e as de um banco B em outro conjunto funcional, e agrupar os dois conjuntos numa classe de conflito. Assim, a partir do momento que um determinado corretor obtiver acesso ao conjunto do banco A, ele não terá acesso a nenhum outro conjunto da mesma classe de conflito (no caso, às in-

[1] Esse bit, associado à aplicações, instrui o sistema operacional a executar o programa em nome do usuário proprietário da aplicação, e não do usuário que a executou

formações pertinentes ao banco B), mas continua tendo acesso a informações em outras classes de conflito. Repare que esse modelo é funcionalmente diferente dos apresentados anteriormente. Enquanto os outros pressupõem regras pré-estabelecidas, neste modelo as restrições vão sendo estabelecidas à medida que o usuário vai acessando às informações contidas nas diversas classes de conflito.

Apesar de prover tanto confidencialidade quanto integridade, o Modelo Muralha da China não é adequado para uso em sistemas operacionais, uma vez que o controle de acesso nesses não é derivado de ações anteriores do usuário, e sim de premissas sobre quais as informações a que ele deve ter acesso. Nesse sentido, os modelos de *DTE* e *RBAC* são mais adequados.

O modelo de *DTE* é derivado do modelo de *TE* 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 [46]. 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 processos em outros domínios.

A linguagem utilizada para a especificação é conhecida como *DTEL* (*Domain and Type Enforcement Language*) e 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 classificada como linguagem de alto nível. Por exemplo, considere uma política de segurança que determine que um navegador somente possa ler seus arquivos de configuração, armazenar arquivos no diretório temporário e executar aplicações específicas para alguns conteúdos da Internet. A Figura 3.1 apresenta a especificação dessa política em *DTEL*, utilizado a notação apresentada em [46].

A primeira linha define os tipos dos arquivos do sistema. A segunda linha especifica o nome do domínio, e qual a aplicação que quando executada entrará nesse domínio. A terceira, quarta e quinta linha definem as regras de acesso do domínio, onde `c,r,w,x,d` definem respectivamente, criação (*create*), leitura (*read*), escrita (*write*) e execução (*execute*) de arquivos e listagem de diretório (*search*). As linhas 6 e 7 definem os arquivos e diretórios associados com cada tipo, onde `-r` indica que tipo deverá ser associado recursivamente nos subdiretórios.

Assim como as demais políticas, o *DTE* combina o controle de acesso mandatório, especificado pela política, com o controle de acesso discricionário. Dessa forma, usando o exemplo anterior, o navegador sendo executado pelo usuário 'diogo' só terá acesso às

```
type t_tmp, t_browser_conf, t_plugin_exec;
domain d_browser = (/usr/bin/mozilla),
                   (crwd -> t_tmp),
                   (rd -> t_browser_conf),
                   (rx -> t_plugin_exec);
assign -r t_tmp /tmp/;
assign -r t_browser_conf /home/{user}/.mozilla/
...
```

Figura 3.1: Exemplo de política escrita em DTEL

configurações contidas no diretório `/home/diogo/.mozilla/`, enquanto outra instância executada pelo usuário 'paulo' só terá acesso ao diretório `/home/paulo/.mozilla/`.

Enquanto o modelo de *DTE* trata a política de segurança em função dos elementos do sistema operacional (usuários, processos e objetos), o modelo de *RBAC* [14, 15] trata o acesso não em função do usuário, mas sim em função do papel do usuário em uma organização. Embora ambos possam especificar o mesmo tipo de segurança em um modelo estático, o segundo trata melhor de movimentação de funcionários entre setores de uma mesma empresa. Por exemplo, considere o caso de uma biblioteca. Um funcionário que atue no setor de empréstimos precisa de acesso de leitura à lista de livros e de acesso de leitura e escrita na lista de empréstimos. Se, em um determinado momento, ele é transferido para o setor de aquisição de livros, em sua nova função ele precisa de acesso de leitura e escrita na lista de livros, mas de nenhum acesso à lista de empréstimos. Quando outro funcionário for alocado para o setor de empréstimos, a ele será disponibilizado o mesmo acesso que o primeiro dispunha. Repare, que nesse caso, a identidade do usuário não influencia nas decisões de segurança, mas sua função sim.

No *RBAC*, portanto, surge o conceito de *papel*, que corresponde ao direito de realizar um conjunto de transações relacionadas a uma atividade específica. Diferente do conceito de domínio do *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. Um detalhe importante do modelo de *RBAC* é que ele define que apenas um papel pode estar ativo por vez e que, ao executar uma aplicação, o processo gerado deverá ser associado ao domínio do papel ativo, e não ao domínio do usuário. Assim, o processo ficará restrito a um subdomínio do domínio do usuário, sendo possível impor o princípio de privilégio mínimo na especificação da política de segurança. Detalhes maiores sobre classificações e padronizações do modelo *RBAC* podem ser encontradas em [40].

Outra característica prevista pelo *RBAC*, derivada do ambiente de organizações, é a

hierarquia de papéis. Dessa forma, pode ser evitada redundância de especificação de permissões entre vários papéis organizados segundo uma hierarquia, simplesmente incluindo o papel imediatamente inferior na hierarquia e definindo apenas as permissões específicas do nível em questão. Na Figura 3.2 é apresentado um exemplo de hierarquia de um hospital universitário, onde as letras `r,w,a` representam, respectivamente, permissões de leitura (*read*), escrita (*write*) e escrita ao final do arquivo (*append*). No esquema proposto, um aluno de medicina pode consultar dados referentes a diagnósticos e aplicação de medicamentos, para uso em trabalhos e pesquisas do curso, porém sem acesso à identidade dos pacientes. Já um enfermeiro, além do acesso já concedido ao aluno, precisa poder adicionar ítens na lista de dosagens, alertando outros enfermeiros e médicos sobre as aplicações de medicamentos já realizadas. Finalmente, o médico pode modificar o registro do paciente, direcionando o tratamento conforme o diagnóstico do paciente. Repare que, além da confidencialidade dos dados pessoais do paciente serem mantidos pelo médico, a lista de dosagens não pode ser corrompida após a inclusão das anotações de aplicações de remédios, garantindo sua integridade.

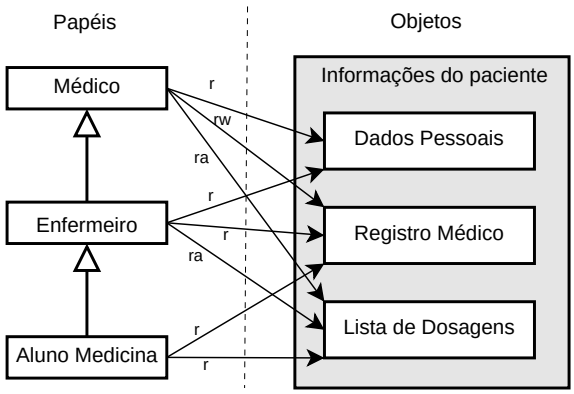


Figura 3.2: Estrutura Hierárquica em RBAC. Exemplo de um hospital universitário.

3.3 Conclusão

Neste capítulo, foram apresentados os principais modelos de políticas de segurança estudados, destacando-se aqueles mais adequados a sistemas operacionais. A grande variedade de modelos existentes deriva das diferenças de requisitos de segurança presentes nos mais diversos ambientes computacionais, o que torna necessário escolher a política a ser utilizada em função dos objetivos de segurança almejados.

A análise das políticas revela algumas características interessantes. As políticas "puras" (só confidencialidade ou só integridade) possuem meios de definir regras de maneira precisa, utilizando elementos matemáticos para definir as regras e o comportamento do

sistema operacional. Porém, normalmente mapeiam permissões em função de usuários e objetos do sistema, o que muitas vezes não condiz com a realidade do sistema operacional ou da organização que o usa. Por sua vez, as políticas híbridas tendem a estar associadas mais ao ambiente que será utilizado do que ao rigor matemático, e são especificas em função dos elementos do sistema operacional (processos e objetos) ou em função de elementos do ambiente de trabalho em que são empregados (papéis ou cargos).

Capítulo 4

SELinux: Sistema e Políticas de Segurança

No intuito de facilitar a adoção de sistemas operacionais seguros, esforços têm sido direcionados para a re-estruturação de sistemas operacionais existentes, objetivando sua adequação às necessidades atuais de segurança, uma vez que os custos de migração para novos sistemas são muito elevados. O sistema operacional Linux, por exemplo, conta com vários projetos para implementar políticas de segurança, como o *Rule Set Based Access Control* (*RSBAC*) [35] , o *Linux Intrusion Detection System*[1] (*LIDS*) e o *Security-Enhanced Linux* (*SELinux*) [28].

O SELinux foi escolhido como base para o desenvolvimento deste projeto por possuir várias características interessantes à segurança do sistema operacional Linux. Ele incorpora ao núcleo do sistema uma arquitetura de controle de acesso mandatório forte e flexível, e provê mecanismos para impor o isolamento de informações baseado em requisitos tanto de confidencialidade quanto de integridade. Sua política de segurança foi implementada para permitir o uso de abstrações dos modelos de *TE* e *RBAC*, sendo possível configurar o sistema para uma grande variedade de objetivos de segurança.

Nesse capítulo, serão apresentados os aspectos do SELinux relevantes à segurança de sistemas operacionais. Primeiramente, será apresentada a arquitetura de segurança do SELinux. Em seguida, serão analisados aspectos da implementação do servidor de segurança, sistema base para a imposição das políticas. Finalmente, será apresentada a política de segurança base do SELinux, utilizada durante o desenvolvimento do projeto proposto nesta dissertação.

[1] `http://www.lids.org`

4.1 Arquitetura

Como o SELinux é uma implementação da arquitetura Flask de segurança no núcleo do sistema operacional, é importante entender primeiramente o que é essa arquitetura. A Flask [43] foi criada na tentativa de servir como uma arquitetura genérica de controle de acesso mandatório (*MAC*). Um importante objetivo desse projeto foi prover um suporte flexível para políticas de segurança, uma vez que nenhum modelo único de *MAC* é capaz de satisfazer a uma ampla variedade de requisitos de segurança. Este objetivo foi alcançado pela separação clara entre a lógica da política de segurança e o mecanismo de imposição da política (*enforcement mechanism*). Cuidados foram tomados para garantir que uma interface de política bem-definida fosse especificada e que pudesse suportar um grande conjunto de políticas de segurança. Adicionalmente, com a arquitetura Flask a imposição de uma política de segurança pode ser transparente para as aplicações pelo fato de permitir a definição de comportamento padrão de segurança, na ausência de regras específicas da política.

O sistema operacional Flask foi implementado com sucesso a partir da adição da arquitetura de segurança proposta ao sistema operacional *Flµke* (abreviação de *Flexible MicroKernel*), apresentando bons resultados com relação aos objetivos iniciais: segurança, flexibilidade de política, baixo impacto no desempenho e escalabilidade [43]. A Figura 4.1 apresenta a arquitetura utilizada para incorporar o Flask no sistema operacional *Flµke*. No intuito de garantir uma adoção maior da arquitetura de segurança, o Flask foi incorporado a um sistema operacional de código aberto largamente utilizado, o Linux, surgindo assim o SELinux (*Security-Enhanced Linux*, ou melhor, Linux com segurança aprimorada).

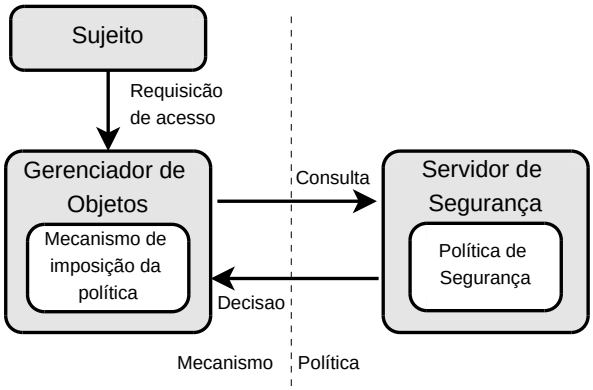


Figura 4.1: Arquitetura Flask

Assim como o sistema operacional Flask, o SELinux conta com um mecanismo de controle de acesso mandatório (*MAC*) integrado aos principais subsistemas do núcleo do Linux, incluindo controles de granularidade fina para as operações sobre processos, arqui-

vos e *sockets*. A lógica de decisão da política de segurança foi encapsulada em um novo componente do núcleo denominado Servidor de Segurança, responsável por tomar decisões referentes a nomeação e acesso dos objetos em resposta a requisições independentes de políticas que foram posicionadas em todo o núcleo. Essa arquitetura permite ao núcleo impor as decisões da política sem necessariamente conhecer os detalhes da política.

O Servidor de Segurança foi desenvolvido com um modelo particular de *MAC* para contornar as limitações de implementações de *MAC* tradicionais. Devido à flexibilidade da arquitetura Flask, entretanto, esse modelo pode ser facilmente modificado, ou mesmo substituído, para suportar outros modelos.

A implementação do servidor de segurança do SELinux demandou a seleção de um modelo concreto de política de segurança, o que levou a escolha de uma combinação de *RBAC* e *TE*. Apesar do SELinux não depender exclusivamente desse modelo, ele foi escolhido por ser suficientemente genérico para suportar muitos dos objetivos de segurança encontrados no dia-a-dia das organizações.

4.2 Servidor de Segurança

O servidor de segurança [28] é totalmente separado do resto do núcleo do SELinux e isolado através de uma interface bem definida. A flexibilidade da arquitetura Flask permite que o servidor de segurança seja modificado ou substituído para alterar os modelos de segurança suportados. O completo encapsulamento de lógica da política de segurança dentro do servidor de segurança torna possível esse requisito da arquitetura sem causar nenhum impacto ao resto do sistema.

As decisões de segurança tomadas pelo servidor de segurança são baseadas em contextos de segurança (*security contexts*) que representam rótulos de segurança associados a todo sujeito e objeto do sistema operacional. Um contexto de segurança é um identificador independente de política que pode ser manipulado por diferentes partes do sistema, mas deve somente ser interpretado pelo servidor de segurança. Esse contexto contém todos os atributos de segurança associados a um objeto rotulado em particular, e que são relevantes à lógica de decisões da política. No caso específico do modelo base adotado, são três os atributos relevantes à segurança armazenados no contexto de segurança: uma identidade, um papel e um tipo.

Entretanto, os contextos de segurança não são ligados diretamente aos objetos. Para garantir a flexibilidade, um segundo identificador independente de política, denominado identificador de segurança (*SID - Security Identifier*) é associado a todo objeto que necessite de rotulação. Os *SIDs* são identificadores locais, não persistentes e opacos, que são mapeados para contextos de segurança. Esse mapeamento é criado em tempo de execução do sistema operacional e mantido pelo servidor de segurança. Quando um objeto é cri-

ado, o servidor de segurança decide qual *SID* deverá ser utilizado como rótulo. Os *SIDs* associados com objetos são passados para o servidor de segurança e usados como base para as decisões de segurança.

Os controles de acesso mandatório do SELinux são implementados como verificações de permissão que foram inseridas em pontos de controle no sistema operacional. Os detalhes de implementação no código do núcleo são apresentados em detalhes em [27]. São aproximadamente 140 permissões de granularidade fina agrupadas em 28 classes de objetos que foram definidas para permitir o controle de praticamente qualquer operação do sistema operacional. Exemplos de permissões incluem 'transição' e 'envio de sinais' para a classe de processos, ou 'criação' e 'escrita' para a classe de arquivos. As permissões de segurança são verificadas em função de verificações feitas entre *SIDs* de origem e *SIDs* de destino para uma permissão particular em alguma classe de objetos. Para responder a uma verificação de decisão, o servidor de segurança utiliza os atributos contidos nos contextos de segurança associados aos *SIDs* de origem e de destino para determinar se uma permissão pode ser concedida.

Nesse ponto, é importante fazer uma breve distinção entre a classe e o tipo de um objeto. A classe está diretamente ligada ao sistema operacional e define a forma como o objeto é tratado. Assim, um objeto pode pertencer a classe 'arquivo', 'arquivo de dispositivo' (*device file*) ou mesmo 'processo', o que define quais as permissões controladas para o objeto. O tipo, por sua vez, é independente da estrutura do sistema operacional, e é utilizado para agrupar vários objetos em um conjunto com características comuns à política de segurança. Assim, 'arquivo de configuração do navegador' e 'arquivos pessoais de usuário' são tipos associados a arquivos e diretórios do sistema de arquivos.

Como pode ser observado, cada verificação de uma operação sobre um par (sujeito, objeto) é custoso em termos de impacto no desempenho do sistema operacional, de tal forma que foi incorporado à arquitetura Flask e, conseqüentemente, ao SELinux um *cache* para as decisões tomadas, denominado *AVC* (*Access Vector Cache*). O *AVC* é um componente do sistema operacional que armazena as decisões já computadas, de forma a minimizar o impacto no desempenho que poderia ser causado pelo servidor de segurança. Como normalmente a realização de uma determinada operação de acesso controlada envolve várias operações subseqüentes sobre o mesmo objeto, o *AVC* armazena, após a primeira consulta, todo o conjunto de decisões relacionado a um sujeito e um objeto, minimizando ainda mais o impacto no desempenho.

4.3 Modelo de Política de Segurança

Como visto anteriormente, o modelo base do SELinux foi baseado em dois modelos de segurança: *RBAC* e *TE*. Nesta seção, será explicada a inter-relação desses dois modelos

para especificar o comportamento geral do sistema operacional.

No modelo do SELinux, além das identidades de usuário e grupo comuns ao Linux (*UIDs* e *GIDs*), todo processo está associado a uma identidade ligada à política de segurança. Mudanças nessa identidade são rigorosamente controladas e a configuração da política permite que apenas certos programas modifiquem sua identidade. Quando o usuário acessa o sistema e apresenta suas credenciais (nome de usuário e senha), a porção referente à identidade no contexto de segurança associado ao *Shell* irá refletir a identidade real do usuário. Repare que a identidade do SELinux não está associada ao *UID* do Linux (exceto quando especificado na política), de tal forma que mesmo que o *UID* de uma processo seja modificado, a identidade SELinux permanecerá a mesma. Além disso, quando um novo programa é executado, a identidade do componente é preservada. Dessa forma, todas as decisões de controle de acesso são baseadas na identidade correta do usuário.

As ações de qualquer usuário são restringidas pela política de *RBAC*. A cada usuário é atribuído um conjunto de papéis que ele pode assumir, e a transição entre papéis é controlada pela política. Apesar de não ser estritamente necessário, a política de segurança padrão foi configurada para limitar a transição de papéis através de aplicações próprias que requerem uma nova autenticação do usuário, de forma a garantir que as transições só ocorram com consentimento do usuário e não pela execução de programas maliciosos.

Os papéis são utilizados para expressar as ações permitidas aos usuários, porém, diferente de como o *RBAC* é descrito em [14], o SELinux define as permissões do usuário para um papel em particular usando a política de *TE*. Enquanto uma política *RBAC* típica iria especificar diretamente as permissões concedidas, a política do SELinux especifica domínios que podem ser adentrados pelos papéis, deixando a atribuição de permissões à configuração *TE*. Apesar de não oferecer nenhuma segurança além daquela que poderia ser obtida por uma política estritamente *TE*, essa forma de *RBAC* permite que a política *TE* seja gerenciada mais facilmente utilizando o conceito de papéis.

Finalmente, o tipo é utilizado para expressar o controle de acesso de granularidade fina sobre todo objeto do sistema operacional, sendo que uma matriz de acesso é definida para determinar as ações permitidas a cada par de tipos. No SELinux, os acessos são expressos em termos de permissões concedidas a cada classe de sujeito e objeto que foram definidas em pontos de controle do núcleo, de forma que todas as permissões precisam ser explicitamente concedidas. Um detalhe importante nesse modelo de política é que não há distinção entre tipos e domínios, diferentemente do que ocorre no modelo apresentando em [46]. Como domínios estão associados somente a processos, e um processo é tanto um sujeito como um objeto para o sistema operacional, o domínio é tratado como qualquer outro tipo, com o único diferencial de que pode ser associado a processos. Reduzindo domínios e tipos a uma única abstração de tipo, uma única tabela pode ser utilizada para expressar tanto a interação entre um sujeito e um objeto quanto a interação entre dois

sujeitos, permitindo, inclusive, a definição de relacionamento entre objetos na política.

Essa forma de *TE* também é distinta no sentido de que as permissões são agrupadas por classes de objetos. O conceito de classe permite definir as permissões sobre o objeto em função dos serviços disponíveis, e permite à política distinguir entre diferentes objetos com o mesmo tipo, por exemplo, concedendo permissões diferentes a um arquivo de dispositivo (*device file*) do que seriam concedidas a um arquivo regular. Isso permite ao SELinux prover permissões com granularidade mais fina do que o tipicamente esperado usando *TE*.

4.4 Linguagem de Especificação da Política

As regras da política de segurança do SELinux são escritas em arquivos de configuração baseados em texto, utilizando uma linguagem desenvolvida a partir dos três modelos de política utilizados. Entretanto, o servidor de segurança não utiliza diretamente essa representação, pois tornaria seu código complexo demais para avaliação de corretude. Ao invés disso, esses arquivos de configuração são verificados e convertidos em uma representação binária mais adequada ao servidor de segurança. Dessa forma, separando-se as representações utilizadas para administração da política e para uso do sistema operacional, garante-se uma boa legibilidade e entendimento das políticas pelo administrador de segurança, e evita-se problemas inerentes à interpretação dessas políticas pelo sistema operacional.

A linguagem da política consiste de um conjunto de definições que podem ser agrupadas nos seguintes conjuntos: Flask, *TE*, *RBAC*, usuários, contextos de segurança e restrições[42].

O conjunto do Flask consiste de um conjunto de definições compartilhadas entre o núcleo do SELinux e a política de segurança. Essas definições estão ligadas às estruturas utilizadas pelo sistema operacional para identificar e validar as operações, e por isso raramente são modificadas. Nesse grupo estão inclusas a definição de classes e seus respectivos vetores de permissão de acesso, e a definição dos *SIDs* iniciais utilizados durante o processo de *boot* do sistema operacional.

As definições da política propriamente dita começam no conjunto de definições do *TE*. Essas definições especificam o nome dos tipos e seus atributos, as regras para transições entre domínios, as regras de acessibilidade e auditoria e asserções. As Figuras 4.2 e 4.3 apresentam a sintaxe das definições de *TE* do SELinux. A definição de tipo `type` especifica o nome, possíveis sinônimos (`alias`) e uma lista de atributos (previamente definidos por `attribute`). Os atributos são utilizados para identificar conjuntos de tipos com propriedades similares, e utilizados para simplificar a especificação de regras. As regras `type_transition` e `type_change` lidam, respectivamente, com transições automáticas

(ou preferenciais) e transições opcionais entre domínios e tipos. Existem dois tipos de transições básicas: transição de um domínio para outro, e transição de tipo na criação de objetos. A transição entre domínios ocorre quando um processo cujo tipo esteja contido em *source_types_set* executa uma aplicação de tipo *target_types_set*, inicializando um novo processo (classe *process*) de tipo *new_type*. A transição de tipos ocorre quando um processo do tipo *source_types_set* tenta criar um novo objeto em um objeto do tipo *target_types_set* (um diretório, por exemplo), associando o tipo *new_type* ao objeto criado. Na ausência dessa regra, se um processo estiver autorizado a criar um novo objeto, ele será criado com o tipo do processo.

```
% Declarações de atributos e tipos de TE
ATRIBUTE attr_name;
TYPE type_name [ ALIAS alias_name_set ], attr1, attr2, ...;

% Regras de Transição de TE
TYPE_TRANSITION source_types_set target_types_set:classes_set new_type;
TYPE_CHANGE source_types_set target_types_set:classes_set new_type;
```

Figura 4.2: Definições do conjunto *TE* da linguagem da política do SELinux (declarações e transições)

As regras de acessibilidade e auditoria definem o conjunto de permissões baseados nos tipos do sistema e a classe do objeto afetado (Figura 4.3). As regras `allow` e `auditallow` definem que um processo do tipo (domínio) *source_types_set* tem as permissões *permissions_set* sobre objetos de classe *classes_set* e tipo *target_types_set*. A diferença entre as duas regras está relacionada ao registro para auditoria de segurança. No SELinux, por padrão, acessos negados são registrados, e acessos permitidos não. Ao usar `auditallow`, está-se instruindo o SELinux a registrar no *log* esse acesso, mesmo que o acesso seja permitido. O caso oposto, não registrar uma negação de acesso, é provido pela regra `dontaudit`.

A última definição de *TE*, a asserção `neverallow`, é utilizada somente durante o processo de compilação da política para sua forma binária. Ela especifica regras de acesso que não deveriam constar na política. Nesse ponto, é importante explicar como os conjuntos (*sets*) são definidos no SELinux. Um conjunto de tipos pode ser especificado agrupando os tipos desejados entre chaves (por exemplo, `{type1 type2 ...}`). Entretanto, os conjuntos podem ser definidos em função de tipos que não fazem parte do conjunto, ou então de tipos que apresentem (ou não) atributos em comum. Assim, o trecho `~{type1 type2}` define um conjunto de todos os tipos do sistema menos os dois citados; o trecho `{attr1 attr2}` define o conjunto de todos os tipos que possuem os atributos citados em comum; e, finalmente, o trecho `{attr -type1 -type2}` define o conjunto de todos os tipos que possuem o atributo citado menos os dois tipos citados (o número de

tipos e atributos em cada definição é variável). Dessa forma, uma asserção impondo que não haja transição de processo para um tipo que não é domínio pode ser especificado da seguinte forma: `neverallow domain ~domain:process transition`, onde *domain* é um atributo comum a todos os domínios, e *transition* é uma permissão associada à classe de objetos *process*.

```
% Regras de Acessibilidade e auditoria em TE
ALLOW sourcetypes_set target_type_set:classes_set permissions_set;
AUDITALLOW source_types_set target_type_set:classes_set permissions_set;
DONTAUDIT source_types_set target_type_set:classes_set permissions_set;

% Asserções a serem validadas durante compilação da política em TE
NEVERALLOW source_types_set target_type_set:classes_set permissions_set;
```

Figura 4.3: Definições do conjunto *TE* da linguagem da política do SELinux (acessibilidade e auditoria)

As definições de *RBAC* são utilizadas para definir os papéis e seus domínios associados, hierarquias e regras de acessibilidade entre papéis. A Figura 4.4 apresenta as sintaxes das definições de *RBAC*. A declaração `role` especifica o nome do papel e o conjunto de domínios aos quais o papel está autorizado. Para definir hierarquias entre papéis, utiliza-se a declaração `dominance`. Finalmente, a transição entre papéis só é permitida se houver uma regra `allow` definindo explicitamente a transição e, ainda assim, a transição só ocorre através de aplicações de autenticação especificas do SELinux com essa finalidade.

```
% Declaração de papéis e hierarquia de papéis do RBAC
ROLE role_name TYPES domain_types_set;
DOMINANCE { ROLE role_name { sub_roles_set } }

% Regras de Transição entre papéis do RBAC
ALLOW current_roles_set new_roles_set;
```

Figura 4.4: Definições do conjunto *RBAC* da linguagem da política do SELinux

Finalmente, as declarações de usuários definem os usuários reconhecidos pela política e especificam o conjunto de papéis autorizados de cada usuário. Apenas os usuários definidos nessas declarações podem ser utilizados na definição de contextos de segurança. A Figura 4.5 apresenta a sintaxe da declaração de usuários.

Com os três conjuntos de definições, especificam-se todos os elementos necessários para compor o contexto de segurança: usuário, papel e tipo. Essa trinca é normalmente representada por uma cadeia de caracteres com a estrutura `usuário:papel:tipo`, e é

```
% Declaração de Usuários
USER user_name ROLES allowed_roles_set;
```

Figura 4.5: Definições do conjunto de usuário e restrições da linguagem da política do SELinux

utilizada pelo SELinux para validar as regras e tomar a decisão de permitir ou não o acesso. Dessa forma, mesmo que haja uma regra *RBAC* permitindo a transição de um papel (origem) para outro papel (destino), um processo do usuário atuando no papel de origem só poderá mudar de papel se o seu usuário estiver autorizado a assumir o papel de destino. Essa hierarquia na tomada de decisões é válida também para as regras *TE*, ou seja, na existência de uma regra que permita a transição de um domínio para outro, a transição só estará disponível a um processo cujo papel esteja autorizado a executar no domínio de destino. Dessa forma, regras mais genéricas e, portanto, mais simples podem ser especificadas no nível de *TE*, sendo possível criar restrições especificas a papéis ou usuários usando as regras desses níveis.

Definidos os três elementos básicos do contexto de segurança, a única etapa que falta para que a política seja funcional é associar todo objeto presente no sistema operacional a um contexto de segurança. Esse mapeamento é feito de duas formas: através de *rótulos persistentes* ou através da definição de *regras de associação*. A primeira forma é aplicada a sistemas de arquivos persistentes que suportem rotulação persistente de segurança, como o `ext3` ou `reiserfs`, e normalmente é inicializada durante a instalação do SELinux ou em caso de modificações da estrutura de tipos da política de segurança. Essas definições ficam armazenadas em um arquivo separado do arquivo de políticas e só precisam ser aplicadas uma vez, já que o sistema de arquivos manterá os rótulos de forma persistente e os novos arquivos criados irão obter seus contextos de segurança a partir das regras da política. A Figura 4.6 apresenta a sintaxe dessas definições, e quatro exemplos, definindo os contextos de segurança de todo o diretório de binários do sistema, do serviço de *login*, dos diretórios do usuário e dos arquivos do usuário, respectivamente. O campo *file_type*, quando não especificado, define que o contexto vale para todas as classes de objeto.

O mapeamento de contextos de segurança através de regras de associação são utilizados em quatro situações onde a rotulação persistente não é possível: definição dos contextos de segurança dos *SIDs* de inicialização do sistema operacional, definição de comportamento de rotulação de sistemas de arquivos, definições de arquivos que não suportar rotulação fixa e definição de contextos para objetos de rede. A Figura 4.7 apresenta a sintaxe dessas definições. A declaração `sid` é utilizada para definir o contexto de segurança dos *SIDs* de inicialização do sistema operacional. As declarações `fs_use_psid`, `fs_use_task` e `fs_use_trans` são utilizadas para determinar o comportamento de rotulação durante a montagem de sistemas de arquivos e criação de arquivos especiais. `fs_use_psid` instrui

```
% Declaração de Contextos de Segurança dos objetos anteriores a política
pathname_regexp ['-'file_type] [security_context]

Ex.:
/bin(|/.*)           system_u:object_r:bin_t
/bin/login           system_u:object_r:login_exec_t
/home/(.*)       -d  user_u:user_r:home_dir_t
/home/(.*)/(.*)      user_u:user_r:user_files_t
```

Figura 4.6: Definições dos contextos de segurança iniciais do SELinux

o SELinux a utilizar o mapeamento persistente ao montar um determinado sistema de arquivos. `fs_use_task` instrui o SELinux a associar aos objetos criados o mesmo contexto do processo que o está criando, e é utilizado para objetos de *pipe* e *socket*. Por fim, `fs_use_trans` instrui o SELinux a derivar o contexto de segurança do objeto tanto do contexto do processo que o está criando quanto o contexto associado ao tipo de sistema de arquivos, e tipicamente ele é usado em pseudo-terminais e memória compartilhada.

```
% Definição de associação inicial entre SID e contexto de segurança
SID sid_id security_context;

% Definições de associações entre sistemas de arquivo e contextos
% de segurança
FS_USE_PSID fstype;
FS_USE_TASK fstype security_context;
FS_USE_TRANS fstype security_context;

% Definição de contexto de segurança para /proc e /dev
GENFSCON fstype path_prefix '-'file_type user:role:type;

% Definições de contexto de segurança para Redes
PORTCON protocol port_range user:role:type;
NETIFCON interface device_context packet_context;
NODECON ipv4_address ipv4_mask security_context;
```

Figura 4.7: Definições dos contextos de segurança dinâmicos do SELinux

Ainda com relação ao mapeamento por regras de associação, existem mais dois conjuntos de definições. A declaração `genfscon` é utilizada para determinar o contexto de segurança em sistemas de arquivos que não suportam rotulação, como é o caso do `proc`, `devfs`,`usbdevfs` e `driverfs`. O último conjunto de definições, `portcon`, `netifcon` e `nodecon` é utilizado para determinar o contexto de segurança de objetos provenientes da

rede, definindo o contexto de segurança de portas, interfaces de redes e endereços IP da Internet, respectivamente.

O último conjunto de definições da linguagem da política do SELinux não influencia no comportamento do sistema operacional, mas sim na validação da política de segurança especificada. Para tal, são definidas restrições sobre permissões, baseadas na forma de expressões booleanas, que precisam ser satisfeitas para que a permissão especificada seja concedida. Detalhes sobre essa estrutura podem ser encontrados em [42].

No intuito de simplificar a especificação da política, o SELinux utiliza um pré-compilador de políticas que permite gerar o arquivo de política para compilação final a partir de uma hierarquia de diretórios e arquivos, permitindo assim agrupar as regras por aplicação e/ou finalidade, e disponibilizando conjuntos de macros para auxiliar a definição da política. A Figura 4.8 apresenta um trecho da macro *domain_trans*, que é utilizada para definir o conjunto de regras comuns à transição para um novo domínio.

```
# domain_trans(parent_domain, program_type, child_domain)
# Permissions for transitioning to a new domain.

define(‘domain_trans’,‘

# Allow the process to transition to the new domain.
allow $1 $3:process transition;

# Allow the process to execute the program.
allow $1 $2:file { read x_file_perms };

# Allow the process to reap the new domain.
allow $3 $1:process sigchld;

# Allow the new domain to write back to the old domain via a pipe.
allow $3 $1:fifo_file rw_file_perms;

# Allow the new domain to be entered via the program.
allow $3 $2:file entrypoint;

...

’)
```

Figura 4.8: Definições (parciais) da macro *domain_trans* do SELinux

4.5 Conclusão

Nesse capítulo, foram apresentados os aspectos do SELinux relevantes à segurança de sistemas operacionais. Sua arquitetura, por ser derivada de um modelo implementado em um sistema operacional de microkernel, possui diversas vantagens sobre outros soluções adaptadas para o sistema operacional Linux, tornando-se um interessante opção como substituto seguro do Linux e seus derivados. Adicione-se a isso o fato de que o isolamento do servidor de segurança do restante do sistema permite uma maior flexibilidade e políticas de segurança, e se obtém um sistema operacional apto a ser utilizado nos mais diversos ambientes computacionais.

A política padrão, definida com base em dois importantes modelos de política de segurança, permite especificar o comportamento do sistema operacional dividindo as regras em camadas de abstração, simplificando não só a especificação da política, mas sua posterior validação e gerenciamento. Finalmente, como ele pode ser instalado em substituição ao Linux, sem necessitar a modificação das aplicações já existentes, influenciam na decisão de utilizá-lo como base para o desenvolvimento da ferramenta de visualização de políticas de segurança.

Capítulo 5

Análise de Requisitos

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 arquivo de especificação da política do SELinux para definir os serviços básicos do sistema possui mais de 7.700 linhas de especificação, não contando comentários e linhas em branco da política. Na tentativa de compreender a política de segurança padrão do SELinux, percebeu-se a necessidade de ferramentas para visualização de políticas que auxiliassem em sua compreensão, o que desencadeou uma pesquisa para identificar as ferramentas disponíveis.

As poucas ferramentas encontradas, e seu foco no formalismo e em sistemas distribuídos, demonstraram a necessidade de desenvolver uma ferramenta que objetivasse auxiliar administradores de políticas de sistemas operacionais na compreensão, análise e verificação das políticas de sistemas operacionais. Com isso em mente, surgiu a idéia de projetar uma ferramenta para visualização gráfica de políticas de segurança, que exibisse de maneira clara e independente do conhecimento da especificação o relacionamento entre usuários, papéis, domínios e tipos do sistema operacional.

Neste capítulo, são apresentadas as atividades realizadas para identificar a necessidade da ferramenta, modelos que poderiam ser utilizados, e quais as questões importantes a serem consideradas no projeto, desenvolvimento e utilização da ferramenta. Primeiramente, são apresentados os trabalhos correlatos estudados. Em seguida, são apresentados os objetivos do sistema e o ambiente de desenvolvimento, que dão base à análise de requisitos apresentada em seguida. Finalmente, apresenta-se o protótipo desenvolvido com o intuito de auxiliar na identificação de estruturas e necessidades do projeto da ferramenta **Policy Viewer**.

5.1 Trabalhos Correlatos

Partindo da necessidade de encontrar ferramentas que auxiliassem na compreensão de políticas de segurança, foram pesquisados estudos e ferramentas que utilizassem representações de políticas não baseadas em arquivos textuais, visto que a linearidade imposta pelo formato textual é um dos fatores que dificultam a análise das políticas.

Com relação à modelagem em estruturas gráficas, foram encontrados três trabalhos envolvendo o mapeamento de políticas de segurança em uma estrutura de grafos, porém com objetivos totalmente distintos dos desta dissertação.

Em [30] é apresentada uma forma de usar grafos para modelar políticas de segurança baseada em contextos, para ser utilizada na especificação da política. Essa modelagem permite que uma política possa ser descrita com maior formalismo do que em outros modelos de políticas baseadas em contexto que utilizam formalismo baseado em regras, com a vantagem adicional de simplificar a visualização da política especificada. Uma característica interessante apontada pelo autor é que a representação em grafo permite uma compreensão melhor da política, auxiliando numa aquisição incremental de conhecimento e na evolução da política através da assimilação de novas práticas de especificação.

Em [22], os autores apresentam uma formalização do modelo de *RBAC* utilizando transformações em grafos. Essa formalização consiste de uma técnica de especificação gráfica que permite a descrição visual intuitiva de estruturas dinâmicas, presentes em modelos de controle de acesso. O uso da estrutura de grafos permite um tratamento uniforme para papéis de usuários e administradores, sem precisar de meta-modelos para descrever possíveis evoluções da estrutura administrativa, e se beneficia de resultados bem estabelecidos em sistemas de transformações em grafos.

O terceiro trabalho analisado baseado em estrutura de grafos é apresentado em [34]. Nesse artigo, é apresentado um modelo de grafo para mapear papéis em hierarquias, que são utilizadas para representar relacionamento entre papéis. Utilizando esse modelo, os autores apresentam uma taxonomia para vários tipos de conflitos, considerando detalhes de conflitos entre privilégios e/ou papéis, e demonstram como o modelo pode ser utilizado para particionar o grafo em coleções não-conflitivas que podem ser seguramente autorizadas a um determinado usuário.

Como pode ser observado, esses três trabalhos utilizam estruturas de grafos para representar a política de segurança, sendo que essas estruturas são utilizadas para obter um maior formalismo na especificação e/ou implementação de estruturas de segurança e para auxiliar na identificação de problemas da política. Apesar de os três artigos citarem ferramentas em desenvolvimento para operação sobre os grafos propostos, pesquisas subseqüentes não encontraram nenhuma ferramenta disponível derivada deles.

Ferramentas cujo objetivo principal é visualização gráfica de políticas foram encon-

tradas na área de sistemas distribuídos. As duas ferramentas identificadas utilizam representações em árvore para mapear os domínios do sistema distribuído. Entretanto, é importante observar que o conceito de domínio em políticas para sistemas distribuídos é um pouco diferente do conceito de domínio para sistemas operacionais. Enquanto neste último um domínio representa o conjunto de permissões associado a um processo ou usuário do sistema, em sistemas distribuídos o domínio representa conjuntos de sujeitos, objetos e recursos computacionais distribuídos pela rede, dessa forma definindo quais sujeitos e objetos podem interagir, e quais computadores da rede eles podem utilizar para realizar o seu processamento.

Em [16] é apresentado um ambiente interativo de gerenciamento de domínios para aplicações empresariais distribuídas utilizando o CORBA, e é baseado na linguagem de configuração de políticas Darwin. Esse ambiente permite a visualização da estrutura de componentes em termos de instâncias de componentes, suas alocações em nós da rede e a conexão entre suas interfaces. Nesse sistema, é utilizada a representação em *treemap* [8] para apresentar a hierarquia de domínios ao usuário. Outro gerenciador, denominado *beagle*, é utilizado para derivar os sub-componentes de cada domínio e suas inter-relações, permitindo a visualização gráfica do estado de configuração de um processo.

Em [12] é apresentado um conjunto de ferramentas integradas para gerenciamento de políticas em sistemas distribuídos, utilizando a linguagem de especificação de políticas Ponder. No sistema proposto, é utilizado o servidor de diretórios LDAP para gerenciar e distribuir os recursos pela rede, com extensões para permitir que objetos pertençam a múltiplos domínios. Das ferramentas apresentadas, apenas uma é interessante com relação aos objetivos desta dissertação. O *Domain Browser* (navegador de domínios) provê uma interface com usuário para gerenciamento de políticas, que consulta as informações necessárias a partir do LDAP e exibe uma visualização gráfica estruturada em árvore representando a relação dos domínios com os objetos. 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, e utiliza um algoritmo de mapeamento em árvores hiperbólicas [8] para disposição dos nós da árvore, o que garante uma melhor navegação na hierarquia.

5.2 Objetivos

A principal motivação para se desenvolver um visualizador gráfico de políticas de segurança é permitir uma melhor compreensão da estrutura de segurança especificada, através da visualização do relacionamento entre os elementos da política de segurança. A representação gráfica de detalhes da política, como a interação entre usuários, papéis e domínios e entre processos e objetos do sistema, permite uma melhor assimilação da especificação

da política do que os arquivos que a especificam. Outro aspecto principal ao projeto é que o uso de representações visuais contribui para a identificação de possíveis riscos de segurança, provenientes de falhas conceituais e/ou de especificação na política, que não seriam identificados através da análise direta da especificação da política pelo administrador de segurança ou através do uso de verificadores de restrições baseado em regras.

Partindo dessa idéia base e da análise dos trabalhos correlatos, foram estabelecidos os seguintes objetivos a serem atingidos pela ferramenta:

- Mapear as regras da política de segurança em uma estrutura de grafo, condizente com a lógica da política;
- Representar a estrutura do grafo gerada utilizando representações visuais;
- Disponibilizar um conjunto de funcionalidades e técnicas que permitam a visualização de determinados aspectos da política, conforme requisitos de segurança comumente analisados;
- Representar os objetos de sistema de forma coordenada com a representação em grafo da política, permitindo a identificação de riscos à segurança do sistema operacional;
- Possibilitar estender a ferramenta para outros modelos de política de segurança;
- Utilizar técnicas de visualização de informação para aprimorar a compreensão da política

5.3 Ambiente de Desenvolvimento

O sistema operacional SELinux foi escolhido como plataforma base para o desenvolvimento da ferramenta. As motivações por trás dessa decisão foram apresentados no Capítulo 4.

Para implementação, decidiu-se pelo uso do paradigma de orientação a objetos, de forma que as linguagens Java e C++ foram cogitadas como linguagem de programação. Ambas possuem um variado conjunto de ferramentas que poderiam ser utilizadas no desenvolvimento, entre as quais destacam-se ferramentas para representação e visualização de grafos, e ferramentas para desenvolvimento de interfaces. Entre ambas, a linguagem C++ foi escolhida por preferência pessoal.

Finalmente, o ambiente de desenvolvimento *Qt Designer*[1] foi escolhido como ambiente de programação, pois disponibiliza um *framework* para desenvolvimento de aplicações em C++. Entre outras coisas, ele permite a integração entre código-fonte e interfaces de forma

[1] `http://www.trolltech.com/products/qt/designer.html`

clara e limpa, disponibiliza um editor de interfaces gráficas e permite o gerenciamento centralizado do projeto.

5.4 Análise

Definidos os objetivos e o ambiente de desenvolvimento, a etapa seguinte consistiu em analisar os requisitos necessários a um visualizador de políticas. A idéia base dessa aplicação é representar a lógica da política em uma estrutura gráfica, e exibi-la ao administrador de políticas—usuário-alvo da ferramenta—de uma forma que lhe seja útil.

O estudo dos trabalhos relacionados demonstrou que mapear as regras da política de segurança em uma estrutura de grafos orientados é viável e pode ser feita de forma direta. Nessa estrutura, os nós representam as abstrações associadas a sujeitos e objetos do sistema (usuários, papéis, domínios e tipos) e as arestas representam regras de acessibilidade, de transição e de dominância. Com isso, uma etapa do desenvolvimento do projeto pode ser simplificada, utilizando bibliotecas para representação dos grafos já existentes. Uma conseqüência interessante desse mapeamento em grafo é a independência entre a representação da política e a especificação, de forma que o projeto poderá ser estendido para outros tipos de política, apenas realizando o mapeamento adequado para a estrutura em grafo.

Exibir o grafo de uma política completa para o usuário não iria contribuir de forma alguma para a compreensão do usuário, dada a quantidade de nós e arestas que derivam da política, e que facilmente tornariam confuso o grafo. Outro detalhe a ser levado em conta é que há necessidade de distinguir nós e arestas com significações diferentes, para permitir uma melhor compreensão dos relacionamentos entre os diversos elementos da política. Nesse sentido, três aspectos foram considerados: representação de nós e arestas com diferentes significados na política, estudo de técnicas de visualização de informação que auxiliassem na representação mais clara da política de segurança e identificação de objetivos de segurança para focar a geração dos subgrafos derivados da política.

Com relação ao aspecto de representações distintas, decidiu-se por utilizar o mesmo grau de abstração que a especificação da política. Assim, nós referentes aos tipos das regras de *TE* seriam representados diferentemente dos nós das regras de *RBAC* e usuários. A representação de contextos de segurança constitui uma questão à parte, pois estes associam os três elementos base das regras anteriores, e estão associados diretamente aos objetos do sistema, sendo por isso representados em uma estrutura a parte. As arestas, por sua vez, seriam diferenciadas em relação a sua finalidade: regra de acesso, regra de transição e relação de dominância. Os detalhes sobre o mapeamento no grafo e suas representações são apresentados na Capítulo 6.

O estudo de técnicas de visualização de informação foi baseado em conhecimentos

prévios e formalizado a partir de consultas a [8]. Entre as técnicas descritas, decidiu-se pelo conjunto mais apropriado para estruturas de grafos: filtragem de dados, *zoom*, exibição de detalhes sob demanda, interação do usuário com a visão representada, entre outros. Muitas dessas técnicas podem ser implementadas utilizando as bibliotecas de representação de grafos, que contam com variado conjunto de funções para manipulação de grafos, como a geração de subgrafos e identificação de características em grafos.

O último aspecto considerado consiste da identificação de objetivos de segurança. Dessa forma, podem-se adicionar funcionalidades à interface do **Policy Viewer** que foquem em determinados detalhes da política, como a identificação de transições de domínio que tornassem a política insegura. Nesse sentido, foram listadas as questões mais comuns utilizadas como premissas na análise e verificação de políticas de segurança:

1. Dado um processo em um determinado domínio, quais outros domínios ele pode atingir?

2. Dados dois domínios, existe alguma forma de um processo no primeiro obter acesso ao segundo?

3. Quais aplicações permitem a transição para um determinado domínio sensível a segurança?

4. Dado um objeto de um determinado tipo, quais domínios podem acessá-lo (para leitura e/ou escrita), direta ou indiretamente?

5. Existe alguma possibilidade que uma informação, acessível somente por um domínio A, possa ser escrita em um arquivo (ou objeto) acessível a outro domínio, quebrando a confidencialidade da informação?

6. A partir da definição dos domínios autorizados a um papel, alguma transição entre domínios está sendo bloqueada ou liberada quando não deveria?

7. Existe o risco de que um usuário, atuando em um determinado papel, utilize-o para corromper outros papéis ou mesmo o sistema como um todo?

Como pode ser observado, as quatro primeiras perguntas podem ser respondidas utilizando algoritmos de percurso em grafos, e permitem identificar objetos expostos e aplicações de risco. A quinta questão aponta para a ocorrência de fluxos de informações no sistema operacional, podendo ser utilizada para identificar falhas na premissa de confidencialidade do sistema, em situações semelhantes à tratada pela propriedade estrela do modelo de Bell-LaPadula. As questões 6 e 7 estão associadas ao comportamento do SELinux com relação às regras da política, em que a maleabilidade do emprego conjunto

dos modelos *RBAC* e *TE* simplifica a definição das regras, mas pode sutilmente ocultar alguns detalhes do comportamento da política.

5.5 Protótipo

Antes de iniciar a etapa de projeto do sistema, foram realizados testes utilizando um protótipo para verificar a viabilidade do mapeamento em grafo e sua utilidade na visualização de políticas. Isso foi feito dado que a transição manual da política de segurança para esboços de grafo é um processo complicado, tendo em vista a complexidade da política (mesmo analisando apenas trechos dela) e a pouca manuseabilidade dos esboços. O desenvolvimento do protótipo foi essencial para o amadurecimento do projeto, contribuindo com a identificação de estruturas de dados necessárias, de funcionalidades relevantes ao sistema e de uma organização mais elaborada para o projeto.

O protótipo foi desenvolvido utilizando pequenas aplicações desenvolvidas na linguagem C++, em conjunto com o gerenciador de banco de dados MySQL[2] e programas visualizadores em Java, utilizadas pela ferramenta Intermap[39][3] do ambiente de educação a distância TelEduc[4].

Cada aplicação foi criada com uma finalidade bem específica, utilizando a base de dados MySQL para armazenar e consultar as regras da política. Por se tratarem de aplicações experimentais, pouco foi feito com relação à simplificação de seu uso para o usuário-alvo (administrador de política), sendo o esforço focado principalmente na funcionalidade e nas estruturas de armazenamento, conforme os objetivos iniciais do desenvolvimento do protótipo.

O gerenciador de banco de dados MySQL foi utilizado para criar e gerenciar as tabelas para representação da política. Ele foi escolhido como repositório por permitir a realização de várias consultas consideradas úteis segundo os objetivos de visualização provenientes da fase de análise, pois permite a filtragem e seleção de elementos em função dos relacionamentos entre tabelas. A Figura 5.1 apresenta o diagrama conceitual de tabelas criado para especificar a base de dados a ser utilizada pelas aplicações do protótipo.

Finalmente, o visualizador de grafos da ferramenta Intermap foi escolhido em função de suas características interessantes ao projeto. Ele disponibiliza uma representação em grafo interativa e possui mecanismos para disposição dos nós e arestas na janela, utilizando o algoritmo *Force Directed* [11] para organizá-los de forma mais clara e coerente. Além disso, por permitir a utilização de arquivos XML para entrada dos dados do grafo (versão mais atual), os grafos gerados a partir da base de dados puderam ser facilmente

[2] `http://www.mysql.com`

[3] A versão da ferramenta utilizada é bem mais recente do que a apresentada no artigo citado.

[4] `http://teleduc.nied.unicamp.br`

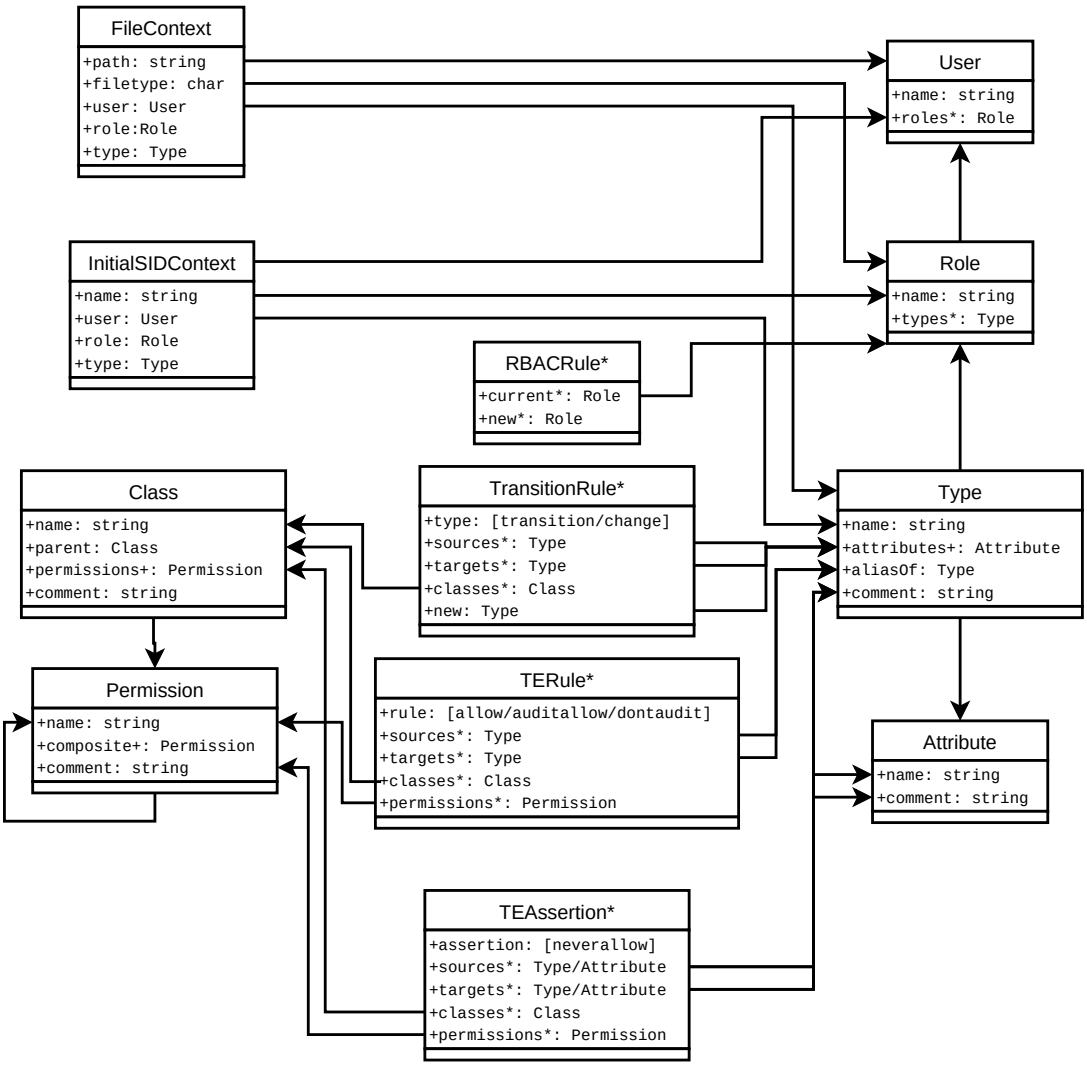


Figura 5.1: Diagrama conceitual das tabelas utilizadas no protótipo.

convertidos em XML para serem visualizados, sendo necessária apenas uma modificação no código do visualizador. Foi adicionada a representação de orientação nas arestas, requisito identificado durante as primeiras visualizações como essencial para a interpretação correta das regras representadas.

Nessa etapa, foram desenvolvidas quatro aplicações que permitiram atingir os objetivos propostos para o protótipo. A aplicação `policyParser` é responsável por ler as regras presentes nos arquivos e inseri-las na base de dados para uso das demais aplicações. Apenas as regras de *TE* da política foram mapeadas nessa etapa. As aplicações `domainRelationshipGraph`, `domainReachGraph` e `domainAllowRules` são responsáveis por acessar a base de dados e gerar os arquivos XML para representação, respectiva-

mente, dos domínios vizinhos (por transição) de um domínio, dos domínios acessíveis (por transição) a partir de um domínio e dos tipos de objetos acessíveis por um domínio. A Figura 5.2 apresenta um grafo gerado pelo visualizador de grafos do Intermap, a partir do arquivo XML gerado pela aplicação `domainReachGraph` tendo como domínio de origem o domínio *user_t*.

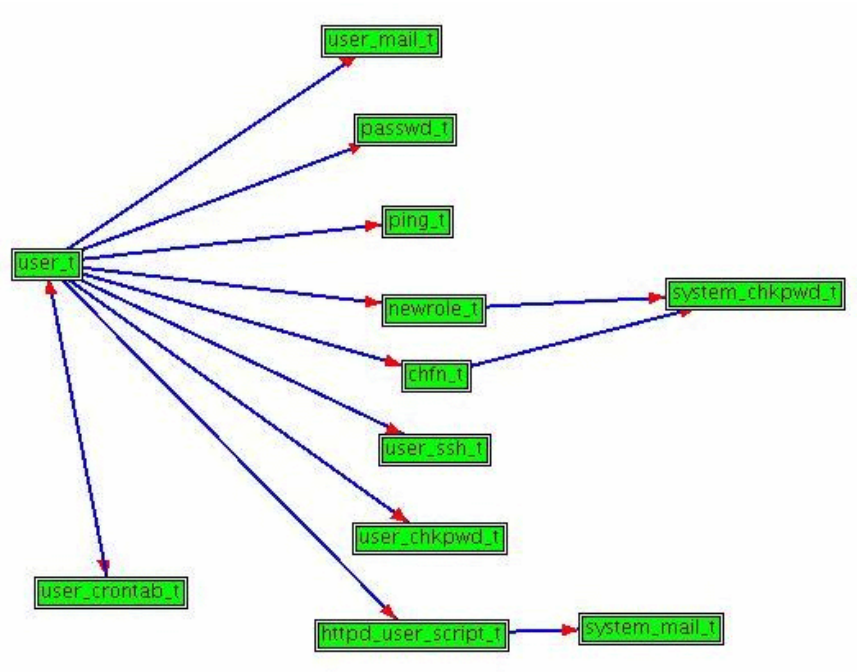


Figura 5.2: Diagrama de Transição de Domínios produzida pelo protótipo. São representados todos os domínios atingíveis a partir do domínio do usuário

Dadas as limitações do visualizador de grafos, alguns detalhes não puderam ser testados, como a utilização de arestas com significações diferentes (por exemplo, regras de transição e regras de acessibilidade). Entretanto, os resultados obtidos durante esses testes preliminares permitiram identificar vários aspectos interessantes ao projeto e à implementação da ferramenta.

5.6 Conclusão

Neste capítulo, foram apresentadas as atividades realizadas para identificar as características necessárias ao projeto. A análise de trabalhos correlatos mostrou a viabilidade do uso de grafos para representação de políticas e que a visualização poderia ser útil para auxiliar o administrador de políticas na compreensão e especificação da política, e também contribuiu com idéias a serem adotadas na ferramenta.

A etapa de análise de requisitos permitiu levantar outras características necessárias à ferramenta, no intuito de facilitar na visualização das políticas, direcionando as representações a serem desenvolvidos em focos específicos de segurança, tornando os grafos mais claros e precisos para o administrador de políticas.

Finalmente, o protótipo contribuiu na identificação de estruturas melhores a serem utilizadas, na seleção do modelo primário de representação a ser elaborado e estendido no projeto, além de servir de uma importante ferramenta de teste para comprovar a utilidade da ferramenta a ser desenvolvida com relação aos objetivos iniciais propostos (visualização, compreensão e análise de políticas de segurança).

Capítulo 6

PolicyViewer

Concluída a etapa de análise e pré-prototipação da ferramenta, iniciou-se a elaboração do projeto a servir de guia no desenvolvimento da ferramenta **Policy Viewer**. Os resultados obtidos nas etapas anteriores foram essenciais na definição das representações, arquitetura e estruturas de dados utilizados na ferramenta.

A partir do projeto, pretendia-se a implementação completa da ferramenta, porém o pouco tempo disponível tornou inviável tal implementação, motivo pelo qual decidiu-se pela implementação parcial da ferramenta, considerando apenas alguns aspectos do projeto, porém de tal forma que fossem possíveis inclusões incrementais para o desenvolvimento posterior de todas as funcionalidade e representações previstas no projeto. A partir da ferramenta desenvolvida, foram elaboradas análises de determinados aspectos da política de segurança padrão do SELinux, para comprovar sua utilidade e eficácia com relação aos seus objetivos.

Neste capítulo, são apresentados o projeto da ferramenta **Policy Viewer**, os detalhes da implementação da ferramenta desenvolvido a partir do projeto e alguns exemplos de uso da ferramenta.

6.1 Projeto

O projeto aqui apresentado corresponde ao projeto final elaborado para a ferramenta para visualização de políticas de segurança **Policy Viewer**, integrando todas as características definidas no protótipo em uma única aplicação. Ele apresenta algumas diferenças em relação ao projeto originalmente desenvolvido devido a características descobertas durante a prototipação e desenvolvimento parcial da ferramenta, que permitiram uma melhor organização e isolamento entre os componentes. Essas diferenças serão explicadas juntamente com o detalhamento da implementação parcial da ferramenta, na Seção 6.3.

A Figura 6.1 apresenta o diagrama conceitual de alto nível dos componentes do sis-

tema. Cada componente é responsável por tratar uma característica da visualização da política em forma de grafo, e essa divisão foi feita para permitir uma desenvolvimento incremental da ferramenta, com a vantagem adicional de permitir inclusões posteriores de representações e mecanismos de exibição ao sistema sem necessitar modificar sua estrutura.

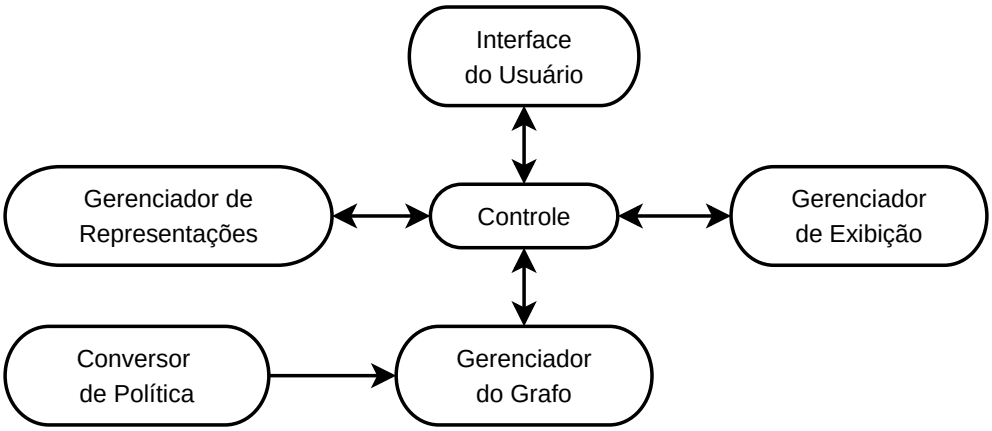


Figura 6.1: Diagrama conceitual do Projeto

Nesse diagrama, o componente *Conversor de Política* é responsável por ler os arquivos de política de segurança do SELinux e converter para a estrutura do grafo utilizada. O componente *Gerenciador do Grafo* é responsável por armazenar a estrutura do grafo e gerar os subgrafos que serão exibidos, a partir de algoritmos de manipulação de grafos. O componente *Gerenciador de Representações* é responsável por gerenciar as representações visuais de grafo, definindo as estruturas de desenho dos nós e arestas a serem exibidos para o usuário. O terceiro componente gerenciador, *Gerenciador de Exibição*, é responsável por gerenciar os algoritmos que cuidam da distribuição visual dos nós e arestas do grafo, de forma coerente e organizada para exibição. O componente *Controle* é responsável por gerenciar a interação entre os três gerenciadores e a interface do usuário. Finalmente, a *Interface do Usuário* é responsável por exibir o grafo ao usuário (no caso, o administrador de políticas) e mediar a interação dele com os elementos da ferramenta.

Para a definição mais formal do projeto, foram utilizados diagramas da linguagem UML[1] (Unified Modeling Language), amplamente utilizada no projeto de softwares cuja implementação faz uso do paradigma de orientação a objetos. A Figura 6.2 apresenta o diagrama de classes elaborado para a implementação do projeto, dividindo as classes conforme previsto no diagrama conceitual.

Como o foco do trabalho é a conversão da política para uma representação em um grafo e a exibição de forma coerente e consistente ao usuário da ferramenta, será dada maior

[1] `http://www.uml.org`

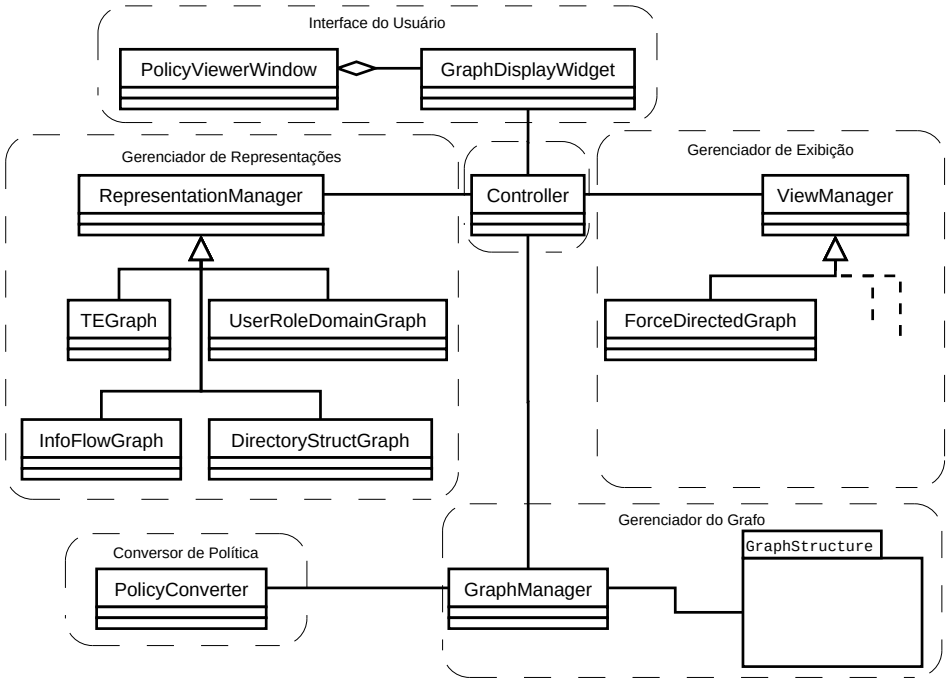


Figura 6.2: Diagrama de Classes do Projeto.

atenção aos componentes *Conversor de Política*, *Gerenciador do Grafo* e *Gerenciador de Representações* e *Interface do Usuário*. Os três primeiros serão detalhados a seguir, e o último será visto em detalhes na apresentação da implementação da ferramenta, na Seção 6.3. Com relação ao *Gerenciador de Exibição*, pretende-se utilizar ferramentas prontas para disposição dos nós do grafo na interface com o usuário, aproveitando muito do que já foi estudado na área de visualização de informação. Exemplos de técnicas desse tipo incluem *Hyperbolic Tree*, utilizado na ferramenta *Domain Browser* apresentada em [12] e o *Force Directed Graph*, utilizado na organização dos nós da ferramenta *Intermap* [11].

6.1.1 Conversor de Política

O conversor de políticas é constituído por uma única classe (*PolicyConverter*), cuja finalidade é ler os arquivos de especificação da política e converter a política para uma estrutura de grafo. Ele foi elaborado primariamente para operar apenas sobre a linguagem de políticas de segurança do SELinux, porém seu isolamento permite realizar futuras extensões para outras linguagens de política, sendo feito o mapeamento adequado dessas novas políticas para a estrutura de grafo utilizada.

A estrutura do grafo a ser gerado é derivada da política da seguinte forma: os elementos das políticas (tipos, domínios, papéis e usuários) são mapeados em nós no grafo. As regras (acessibilidade, transição, autorização) são mapeadas em arestas orientadas, em que a

origem da aresta é definida de acordo com a regra que representa. Assim, em uma regra de acessibilidade, a origem é o tipo (domínio) do processo; em uma regra de transição, a origem é o domínio de origem; e em uma regra de autorização, o tipo é o papel (ou usuário) autorizado a um domínio (ou papel). Os contextos de segurança não são mapeados no grafo, pois são utilizados apenas para representações associadas a objetos dinâmicos do sistema.

É importante notar que uma mesma regra pode gerar várias arestas, visto que podem ser definidos conjuntos de origens e destinos em uma mesma regra, o que seria convertido em várias arestas no grafo. Outro detalhe a ser levado em conta é a necessidade de realizar a transformação da política de segurança do SELinux em duas etapas, dada a forma como são representados conjuntos nas regras. A possibilidade de utilizar conjuntos (`{type1 type2 ...}`), conjuntos complementares (`~{type1 type2}`) e identificar conjuntos de tipos a partir de atributos em comum (`~{type1 type2}` e `{attr -type1 -type2}`) tornam necessário primeiro identificar todos os tipos e atributos existentes, para só então processar as regras e identificar os tipos associados a elas.

6.1.2 Gerenciador de Grafo

O Gerenciador de Grafo é responsável por armazenar a estrutura da política em grafo no sistema, e prover ao componente de Controle um conjunto de funcionalidades para geração de subgrafos a serem utilizadas pelo sistema. Ele substitui o MySQL como centralizador das regras da política representadas em estrutura grafo, que foi descartado do projeto por vários motivos. Entre os motivos, destacam-se o interesse de independência de plataforma da aplicação, facilitando sua migração para outros sistemas operacionais e outras políticas, e o impacto no desempenho de consultas sucessivas e complexas através da interface de acesso ao MySQL, conforme constatado em algumas experiência com o protótipo.

A classe *GraphManager* é responsável por receber os dados do grafo do componente de conversão de política, armazená-la, e gerar os subgrafos a serem exibidos ao usuário. Para geração do subgrafo, foram identificadas as seguintes funcionalidades essenciais: filtragem dos nós e arestas a partir de características destes (atributos de tipos, classes de regras, entre outros), percursos no grafo orientado e identificação de caminhos entre nós. Outros algoritmos poderão vir a ser acrescentados futuramente para suprir necessidades identificadas durante o uso.

O grafo é armazenado na estrutura de classes apresentada na Figura 6.3. É utilizada uma estrutura de grafo genérica, com nós e arestas, das quais são derivadas as classes de nós e arestas específicos da estrutura da política. Nessas classes derivadas são armazenadas não só os nomes e regras em si, mas também informações referentes à origem do elemento na política (arquivo, número de linha, a regra no formato da especificação

original). Os nós *TypeNode*, *RoleNode* e *UserNode* representam, respectivamente, tipos, papéis e usuários da política. É importante ressaltar que, assim como no SELinux, os domínios na representação em grafo também são representados por tipos. O quarto tipo de nó, *ObjectNode*, é utilizado para representar elementos do sistema operacional (arquivos, conexões).

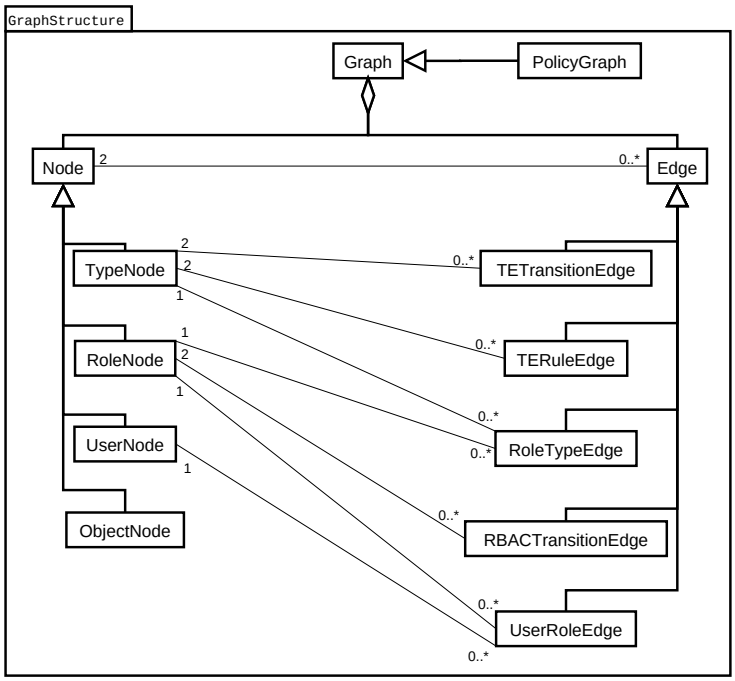


Figura 6.3: Diagrama das classes de dados da estrutura do grafo.

As arestas *TERuleEdge* representam as regras de acessibilidade de *DTE* (`allow` e `auditallow`). A regra `dontaudit`, por instruir o SELinux apenas a não auditar uma negação de acesso, e portanto não representar uma permissão, não é mapeada no grafo. As arestas *TETransitionEdge* representam regras de transição entre domínios e tipos. Além de armazenar o domínio de origem e o tipo (ou domínio gerado), elas armazenam também o tipo do elemento de transição (aplicações, para transição de domínios, e objetos *container*, para transição de tipo). As arestas *UserRoleEdge* e *RoleTypeEdge* representam as regras de autorização para associação entre usuários e papéis, e entre papéis e tipos (domínios). Finalmente, as arestas *RBACTransitionEdge* representam as regras de transição entre domínios. Nesse caso, diferentemente do que ocorre com *TETransitionEdge*, não há necessidade de armazenar informações sobre os elementos de transição, pois o SELinux determina que transições de papéis sejam realizadas apenas por aplicações específicas que utilizam autenticação para validar a transição.

6.2 Gerenciador de Representações

O Gerenciador de Representações é o componente mais importante do sistema, uma vez que ele centraliza e gerencia as representações visuais da estrutura do grafo para o usuário. Nesta seção, serão descritas as funcionalidades gerais das classes de representação e serão apresentadas as representações visuais elaboradas para auxiliar na compreensão e análise das políticas.

A classe *RepresentationManager* é uma classe genérica para manipulação das propriedades gráficas do grafo, da qual são derivadas as classes responsáveis por cada representação. Ela obtém a estrutura do grafo (nós e arestas) do componente Gerenciador de Grafos, define as propriedades gráficas (forma e cor de representação dos nós, estilo das arestas), e interage com os componentes Gerenciador de Exibição (para obter a disposição dos nós) e Interface do Usuário (para o desenho na janela de exibição e interação do usuário). Toda essa interação entre componentes é realizada através do componente Controle, permitindo um melhor isolamento das definições de representação dos demais componentes do sistema.

Para a representação das políticas, foram elaborados quatro diagramas que podem ser compostos para gerar um único grande diagrama da política, porém foram especificados separadamente para permitir a visualização individual das características da política. Dada a estrutura utilizada, outros diagramas podem ser elaborados e facilmente incluídos no sistema, sem requerer modificações nos demais componentes do sistema, simplificando o processo de extensão do **Policy Viewer** a outras visualizações e políticas.

Com relação à política, é importante frisar que apenas regras relacionadas à concessão de permissões foram mapeadas para a estrutura do grafo. Como visto no Capítulo 4, a permissão de realizar uma ação só é concedida se explicitamente definida pela política. Regras referentes a asserções e restrições são verificadas somente no momento da compilação da política para identificar erros na especificação, não afetando o comportamento do sistema após a ativação da política. Adicionalmente, o mapeamento das restrições e asserções na estrutura do grafo demandariam o uso de uma novo tipo de aresta, cuja semântica seria inversa à das arestas utilizadas para representar regras de permissões, isto é, representariam arestas (regras) que não deveriam existir no grafo. Com isso, a representação poderia gerar uma interpretação dúbia da política, de forma que optou-se por não representar essas regras.

Um último detalhe a ser observado, antes de descrever as representações propriamente ditas, está relacionado à coloração dos nós. Derivada da necessidade de distinguir tipo e domínio (que são, para o SELinux, ambos tipos), optou-se por implementar uma característica adicional às representações: além de contarem com cores previamente definidas para cada elemento do modelo, os usuários poderão configurar o sistema para destacar nós

e/ou arestas baseados em suas características. Por exemplo, o usuário poderia utilizar os atributos dos tipos para destacar domínios administrativos de domínios comuns, ou então destacar todos os tipos associados a sistemas de arquivos. Dessa forma, pode-se destacar os pontos de interesse da política, simplificando seu processo de análise.

As próximas subseções descrevem os quatro diagramas desenvolvidos no intuito de permitir a visualização gráfica das políticas.

6.2.1 Diagrama de Domínios e Tipos

O primeiro diagrama apresentado é baseado no *Domain Browser* descrito em [12], porém aplicado à política do sistema operacional e não de um sistema distribuído. Além das transições de domínios exibidas pelo *Domain Browser*, este diagrama apresenta transições de tipo e as regras de acessibilidade.

O SELinux utiliza o mesmo par de regras para definir transição de domínios e transição de tipos. Porém, essas transições são semanticamente diferentes e, portanto, sua representação foi diferenciada. A primeira representação a ser analisada é a de *transição de domínios*. No SELinux, um processo não pode trocar de domínio após ser instanciado, de forma que a transição de domínios só é possível quando um novo processo é criado a partir da execução de uma aplicação. O domínio do processo criado é determinado em função do domínio do processo que o criou, e do tipo da aplicação executada. Entretanto, existem situações em que um mesmo par (domínio origem, tipo da aplicação) pode levar a mais de um domínio—sendo um preferencial (ou obrigatório) e os demais opcionais—tornando-se necessário distinguir essas duas possibilidades no diagrama. A Figura 6.4 apresenta a representação utilizada para a transição preferencial (a) e opcional (b). Repare que essas representações exibem apenas o domínio de origem e o domínio de destino da transição, o que torna necessário a inclusão de um nó "fantasma" para destacar o tipo da aplicação envolvida na transição (c). Essa última forma de exibição é opcional, ficando a critério do usuário determinar se deseja ou não visualizar essa informação.

A segunda forma de representação a ser analisada é a *transição de tipos*. Diferente do que ocorre na transição de domínios, a transição de tipos ocorre quando um processo cria um novo objeto dentro de um objeto *container*, como um diretório ou interface de rede. As regras de transição de tipo definem, portanto, qual o tipo do objeto criado em função do tipo do objeto *container* e do domínio do processo. Durante os esboços das representações, foi observado que o grande problema dessa representação era manter a semântica sem poluir demais o diagrama, caso contrário o usuário poderia ficar confuso com relação à regra apresentada. Após algumas tentativas, decidiu-se por utilizar a representação exibida na Figura 6.5. Assim como no caso da transição de domínios, a transição de tipos pode ser preferencial (a) ou opcional (b), e o usuário pode desejar ver

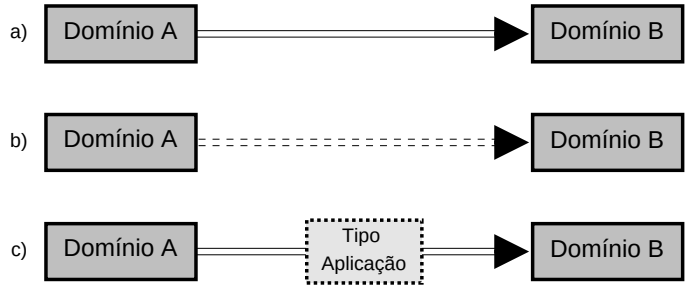


Figura 6.4: Diagrama de Domínios e Tipos: Regras de Transição de Domínio. (a) transição de domínio obrigatório. (b) transição de domínio opcional. (c) transição de domínios com destaque para o elemento de transição.

em destaque o tipo do objeto *container* (c). Repare que, apesar de ambos utilizarem o retângulo como identificador (lembrando que o SELinux trata domínios como tipos especiais), foram usadas cores diferentes para representar cada um (no caso da figura, tons de cinza diferentes).

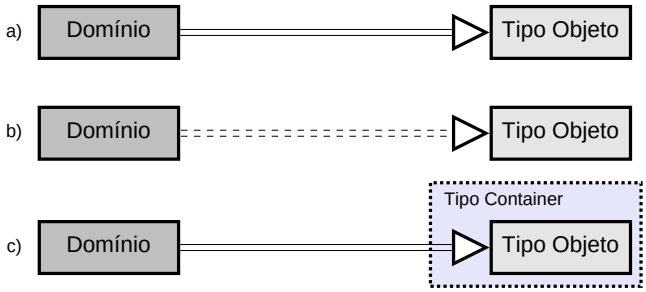
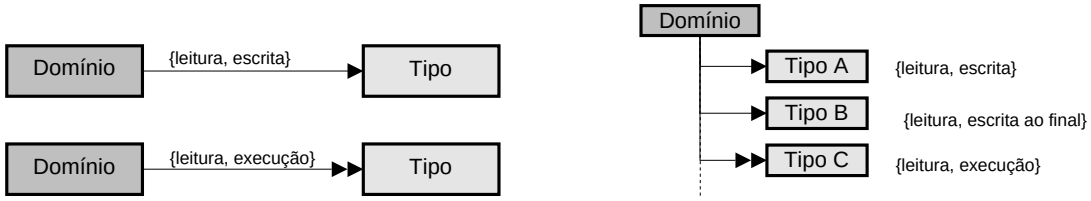


Figura 6.5: Diagrama de Domínios e Tipos: Regras de Transição de Tipo. (a) transição de tipo preferencial. (b) transição de tipo opcional. (c), transição de tipo com destaque para o tipo do objeto *container*.

O último conjunto de regras a ser representado neste diagrama são as *regras de acessibilidade* da política do SELinux. Existem duas regras que determinam quais as permissões de um domínio sobre um objeto: `allow` e `auditallow`. Para os processos associados aos domínios, não há distinção entre uma ou outra, pois ambas determinam a possibilidade de executar uma ação. Entretanto a segunda regra instrui o SELinux a registrar a realização da ação, que normalmente seria feita somente em caso de negação. Por isso, as duas regras são diferenciadas, conforme apresentado na Figura 6.6(a).

Um detalhe observado durante os esboços é que, em determinadas situações, a visualização das regras de acessibilidade em conjunto com as regras de transição tornam o grafo muito poluído, podendo obscurecer detalhes mais importantes da política. Com isso em mente, foi elaborada uma segunda forma de representação das regras de acessibilidade,



(a) para exibição no mesmo diagrama de transição de domínios. As permissões associadas são exibidas sobre a aresta

(b) para exibição a parte com foco em um domínio. As permissões são exibidas ao lado dos tipos

Figura 6.6: Diagramas de Domínio e Tipos: Regras de Acessibilidade. As arestas com seta dupla indicam permissões auditadas.

para ser utilizada na análise de domínios isolados, que é apresentada na Figura 6.6(b). Essa representação é mais adequada na análise de regra para um único domínio, enquanto a anterior é mais apropriada para análise de impacto de segurança indireto, isto é, através da transição sucessiva de domínios.

Em conjunto com as opções de filtragem e manipulação de grafos provida pelo componente Gerenciador do Grafo, esse diagrama pode ser utilizado para analisar, em detalhes, praticamente todas regras de *TE* da política do SELinux. Um detalhe a ser observado é que, assim como ocorreu com as asserções e restrições, a regra `dontaudit` não é mapeada no grafo, uma vez que não influencia no comportamento do sistema, pois o acesso seria negado mesmo na sua ausência. A finalidade dessa regra é instruir o SELinux a não registrar no *log* do sistema uma negação de acesso que não seja importante à segurança do sistema. não consistindo assim uma permissão e, por conseguinte, uma aresta de permissão na estrutura do grafo.

O último detalhe pendente com relação a esse diagrama está associada à característica mista da política de segurança SELinux, que se reflete nas regras de transição de domínio. As regras de transição são limitadas pelos domínios autorizados ao papel associado ao processo. Assim, mesmo que exista uma regra de transição de um domínio para outro, a transição não poderá ser efetivada se o papel associado ao processo não estiver autorizado a assumir o segundo domínio. Assim, tem-se um domínio acessível pela regra de transição, porém inacessível devido às restrições impostas ao papel associado. Para representar essa situação, utiliza-se a representação apresentada na Figura 6.7. No modelo apresentado, um usuário acessando o sistema remotamente (via ssh) é associado ao papel *remote_user_r*. As regras de transição permitem a transição do usuário para os domínios *user_mail_t*, *user_cron_t* e *sysadm_t*, porém o papel *remote_user_r* não é autorizado a entrar no domínio *sysadm_t*, impossibilitando assim a transição.

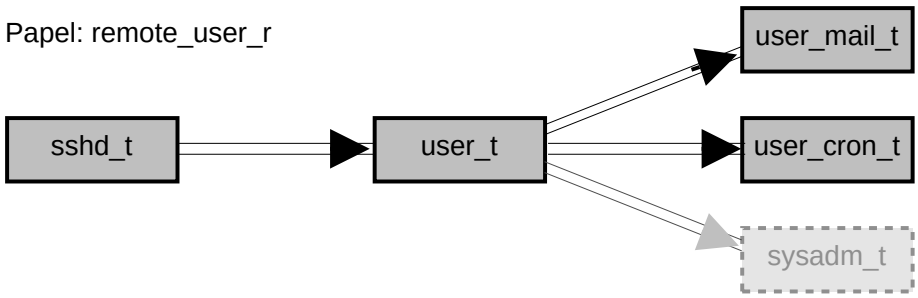


Figura 6.7: Exemplo de diagrama de transição de domínios, analisando o comportamento sob um determinado papel.

6.2.2 Diagrama de Fluxo de Informações

Apesar de útil na visualização de associações entre domínios, permissões e tipos, o diagrama anterior não permite a identificação de caminhos de fluxo de informação. Ao tratar de confidencialidade, o acesso direto de leitura a informações não-autorizadas é apenas um dos pontos de risco na quebra de segurança. A possibilidade de obter o acesso a informações indiretamente, através da utilização de arquivos como túneis de informações entre domínios, representa um risco muito maior, pois a identificação de uma falha desse tipo na política de segurança é muito mais complicado.

Com isso em mente, foi projetado um diagrama de fluxo de informações para representação da política de segurança, utilizando como base a estrutura já disponível pelo componente Gerenciador do Grafo. Diagramas de fluxo de informação já foram usados com sucesso em outras áreas, como engenharia de software e redes, o que demonstra sua utilidade também para o caso de políticas de segurança de sistemas operacionais.

Partindo-se do modelo do diagrama de Domínios e Tipos, apenas duas modificações principais precisam ser feitas para adicionar ao grafo a semântica de caminhos de fluxo de informações: apenas permissões de leitura e escrita em objetos são consideradas e a orientação das arestas segundo o modelo original são modificadas. A modificação da orientação das arestas é feita de forma a indicar o fluxo da informação entre domínios e tipos, de forma que as arestas representando permissão de leitura têm sua orientação invertida, indicando que a informação flui do objeto associado ao tipo para o processo associado ao domínio. Se a aresta representar uma regra que possua tanto permissão de leitura quando de escrita, a aresta é desenhada como bidirecional. A Figura 6.8 apresenta um paralelo entre as arestas de acessibilidade dos dois diagramas.

A Figura 6.9 apresenta um exemplo do diagrama de fluxo de informações, utilizando uma política fictícia. Repare que, segundo a política representada, um processo executando no domínio *user_t* não pode acessar as informações associadas ao tipo *secret_data_t*. Porém, através do domínio do navegador Web (*mozilla_t*), ele pode ter acesso ao diretório

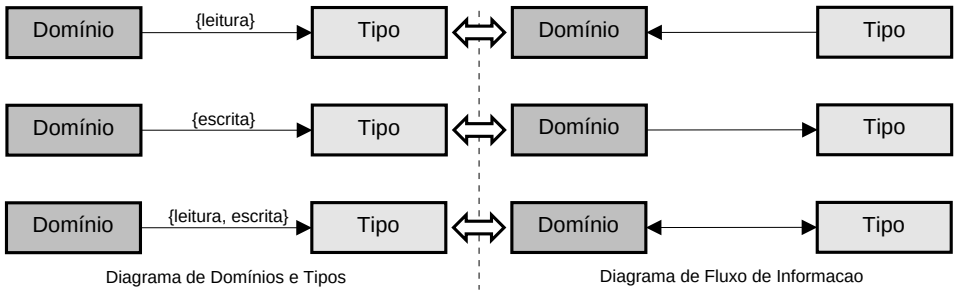


Figura 6.8: Paralelo entre o Diagrama de Domínios e Tipos e o Diagrama de Fluxo de Informações.

temporário, e transferir eventuais "lixos" de processos executando no domínio *secure_t*, e transferi-los para a *cache* do navegador, podendo acessá-los posteriormente. Uma análise direta na especificação da política dificilmente identificaria essa possibilidade de quebra de confidencialidade, e pela análise pode-se restringir a possibilidade de acesso pela diferenciação dos tipos dos arquivos temporários do domínio *secure_t* e *mozilla_t*.

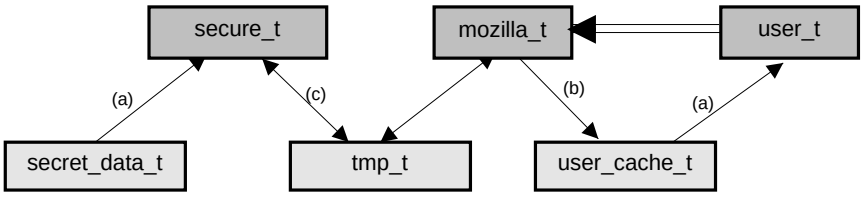


Figura 6.9: Exemplo de diagrama de fluxo de informação no SELinux. As arestas marcadas com a, b, c, representam, respectivamente, permissões de leitura, de escrita, e de leitura e escrita.

Usados em conjunto, os diagramas de Domínios e Tipos e de Fluxo de Informação podem auxiliar na identificação de muitas características da política de segurança que são obscurecidas pela forma textual e linear da linguagem de especificação da política. Isso vale não só para o SELinux, mas para praticamente qualquer política que possa ser mapeada na estrutura de grafo proposta.

6.2.3 Diagrama de Usuários e Papéis

Até o momento, foram tratados apenas detalhes do comportamento do sistema operacional baseado em tipos do SELinux. Por estarem ligados diretamente aos elementos do sistema operacional, os tipos têm importante papel na definição da interação entre os elementos, sendo o modelo mais complexo da política do SELinux.

Para apresentar os outros dois elementos da trinca do contexto de segurança, papéis e usuários, utilizou-se uma representação visual diferente, aproveitando-se da semântica da

estrutura de representação UML. A idéia é utilizar o conceito de agregação para representar as associações entre papéis e domínios, papéis e papéis, e papéis e usuários. Apesar dessa distinção na forma de representação, a estrutura de dados interna do **Policy Viewer** para representar essas estruturas corresponde a um grafo orientado descrita na Seção 6.1.2. Essa representação é utilizada para destacar o relacionamento de autorização entre esses elementos. A Figura 6.10 apresenta uma política fictícia, com 2 usuários, 3 papéis e 3 domínios. Conforme pode ser observado na figura, o papel P está autorizado a utilizar os domínios A e B, enquanto que o papel Q está autorizado a utilizar o domínio C e os domínios autorizados ao papel P. A significação do modelo para usuários, representada em (b), segue a mesma lógica.

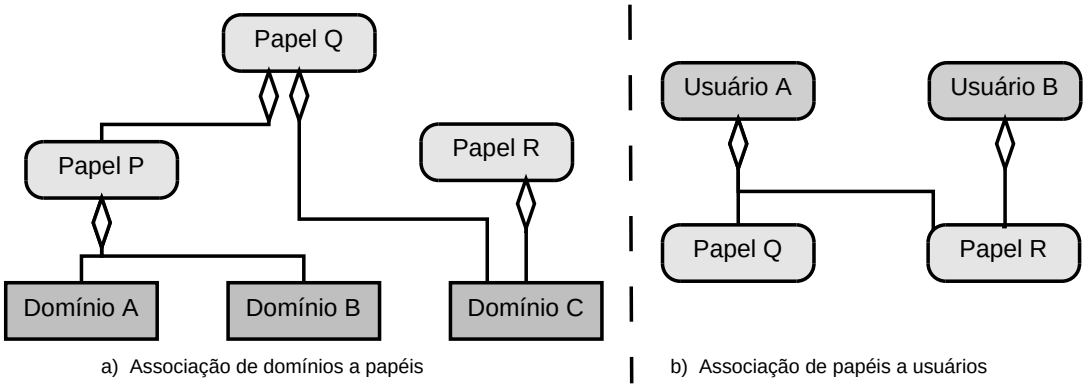


Figura 6.10: Exemplo de Diagrama de Usuários e Hierarquia de Papéis.

Os três diagramas apresentados até aqui mapeiam todas as características estáticas do comportamento do sistema operacional segundo a política de segurança especificada. É fácil observar que este último diagrama pode ser combinado com um dos dois anteriores para representar, em um único grafo, o comportamento dos vários níveis da especificação do SELinux. Entretanto, dado o número de elementos envolvidos, a separação em diagramas distintos permite ter uma visão mais clara dos detalhes de cada nível, e a possibilidade de visualizar os diagramas lado a lado pode facilitar ainda mais a interação do usuário do **Policy Viewer** com a política.

Associado a recursos de filtragem e manipulação de grafos e dos mecanismos de disposição dos nós dos outros dois componentes de gerenciamento do sistema, tem-se uma importante ferramenta para análise e compreensão da política em modo gráfico.

6.2.4 Diagrama de Objetos do Sistema Operacional

Finalmente, após o mapeamento completo das definições estáticas da política de segurança, é necessário mapear os elementos dinâmicos na política. Basicamente, este mape-

amento está relacionado aos contextos de segurança associados aos objetos do sistema, e é resultado das regras de associação da política SELinux e do uso de *SIDs* persistentes pelos sistemas de arquivos.

Entretanto, ao contrário do que ocorre com as regras estáticas, utilizar a estrutura de um grafo para representá-los seria inviável e pouco útil no intuito de auxiliar na visualização da política, por dois motivos. Primeiramente, o número de objetos do sistema é muito grande (mesmo que o grupo demonstrado seja sujeito a filtragens), em segundo lugar o contexto de segurança está associado a três elementos dos diagramas anteriores, e não a um só. Por isso, adotou-se a representação em árvore comumente usada para mostrar a estrutura de diretórios, adicionando-se os detalhes referentes à política de segurança, conforme apresentado na Figura 6.11. Uma vez que o SELinux, assim como o Linux, mapeia todos os recursos do sistema em arquivos especiais do sistema de arquivos, essa representação permite exibir todos os objetos do sistema sem requerer a criação de um diagrama diferenciado. Dessa forma, simplifica-se a interação do usuário com o **Policy Viewer**, já que ele já estará acostumado a utilizar essa estrutura.

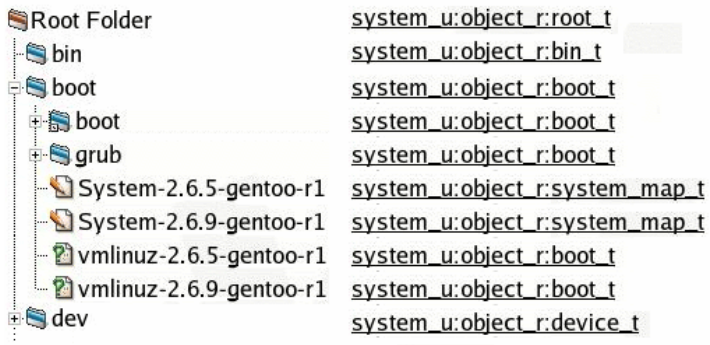


Figura 6.11: Diagrama de Objetos do Sistema Operacional.

Na representação, os objetos do sistema podem ser filtrados levando em conta características da política de segurança. Por exemplo, poderiam ser apresentados todos os objetos acessíveis para leitura por um determinado domínio, para verificação de aspectos de confidencialidade associados ao domínio. Ou então, poderiam ser exibidos os objetos acessíveis direta ou indiretamente a partir de um determinado domínio, permitindo verificar se há riscos a integridade do sistema operacional.

6.3 Policy Viewer: detalhes da implementação

A ferramenta foi implementada seguindo a estrutura prevista pelo projeto, porém algumas experiências obtidas durante o desenvolvimento da ferramenta serviram para realimentar

o projeto e algumas modificações no mesmo foram derivadas de necessidades identificadas durante a implementação. Alguns nomes de classes e algumas estruturas diferem entre o projeto e a ferramenta devido a essa realimentação, porém, tendo em vista o isolamento entre componentes obtido, a modificação para adaptação ao projeto pode ser realizada sem grandes impactos ao código.

Para implementação parcial da ferramenta, foram escolhidos um conjunto de regras a serem mapeados e um conjunto de funcionalidades essenciais relacionadas a estas regras. Decidiu-se por iniciar a implementação da ferramenta pelas regras relacionadas ao modelo *TE* previsto pelo SELinux, que correspondem ao conjunto das regras de transição de domínios e tipos, e ao conjunto de regras de acessibilidade, pelo fato de definirem o comportamento do sistema operacional em função dos elementos reconhecidos por ele: processos e objetos. Entretanto, alguns contratempos durante a etapa de desenvolvimento permitiram somente a implementação das características do diagrama relacionadas às transições, e as regras de acessibilidade foram deixadas para implementação posterior, quando o projeto for totalmente implementado.

Com relação ao projeto apresentado na seção anterior, as seguintes classes foram implementadas:

- *PolicyConverter*: Implementação parcial da estrutura prevista, realizando o *parsing* das regras de transição de domínios e tipos. Diferente do projeto final, entretanto, os métodos dessa classe foram implementadas na classe *PolicyGraph*.

- *GraphManager*: Implementação parcial, sendo responsável por gerar os subgrafos a serem exibidos para o usuário do **Policy Viewer**. Como anteriormente cada representação estaria associada às rotinas de geração de subgrafo, os métodos foram implementados inicialmente na classe *DomainTransitionGraph*.

- *GraphStructure*: Implementação parcial, mapeando as regras de transição somente.

- *Representation Manager* e *TEGraph*: Implementação completa, porém em uma única classe, denominada *DomainTransitionGraph*. Foi durante a implementação dessa classe que observou-se a necessidade de reestruturar o projeto para torná-lo mais simples e prático. Repare que, primariamente, previa-se o uso de dois diagramas distintos para representação de transições e para representação das regras de acessibilidade, do que deriva o nome. Entretanto, o uso de regras de filtragem mostrou-se mais útil do que esta divisão, permitindo uma melhor estruturação da representação.

- *GraphDisplayWidget*: Implementação completa. Os métodos disponíveis nesta classe permitem a manipulação dos nós na representação (arraste, seleção, adição e remoção de nós), conforme será apresentado na Seção 6.3.1.

- *PolicyViewerWindow*: Implementação parcial, cumprindo os objetivos relacionados as regras de transição. Ao contrário da estrutura apresentada no diagrama de classes, esse "módulo" é composto por várias subclasses responsáveis pelos aspectos da interface com o usuário (menus, menus de contexto, botões, entre outros). Sua estrutura está baseada nos padrões de implementação de interfaces do *QtDesigner*.

- *Controller*: Implementado na classe *DomainTransitionGraph*. O projeto inicial não previa uma classe de controle, tendo como elemento central o componente responsável pelas representações. A possibilidade de criar extensões futuras motivou a separação no projeto desse componente.

- *ViewManager*: Não implementado. As rotinas de geração de grafo utilizam números aleatórios para realizar o posicionamento inicial dos nós na interface do usuário.

Como pode ser observado, o desenvolvimento preliminar do projeto permitiu identificar várias características do projeto que poderiam ser modificadas para criar uma estrutura mais organizada, permitindo expansões posteriores no projeto. A seguir, são apresentadas as características da interface desenvolvida para a ferramenta, e alguns detalhes a serem adicionados à versão final.

6.3.1 Interface com o usuário

A interface com o usuário foi projetada visando a simplicidade de interação do usuário, porém provendo o maior conjunto de recursos possíveis para auxiliar o usuário na visualização e análise da política de segurança. A Figura 6.12 apresenta a interface básica com o usuário. Estão destacados na figura: (a) menu principal; (b) descrição do grafo, (c) *GraphDisplayWidget*; (d) nós; (e) arestas; (f) *hint*.

O menu principal permite ao usuário abrir os arquivos de política, aplicar opções de edição sobre o grafo visualizado e escolher quais as visualizações desejadas. Na implementação parcial, o usuário conta com três opções de visualização de transição de domínios, que correspondem a três preocupações de segurança (Figura 6.13). Essas opções permitem verificar as 3 primeiras questões apresentadas na Seção 5.4, ou seja, determinar se um determinado domínio pode atingir algum domínio de risco, se um determinado domínio está exposto a algum domínio que não deveria e se existe um caminho entre dois domínios que transições sucessivas poderiam permitir a transição do primeiro para o segundo.

A descrição do grafo está presente para orientar o usuário de qual o conjunto de regras estão sendo analisados, visto que o usuário pode abrir mais de uma janela para visualizar aspectos distintos da política de segurança. No exemplo da Figura 6.12, a

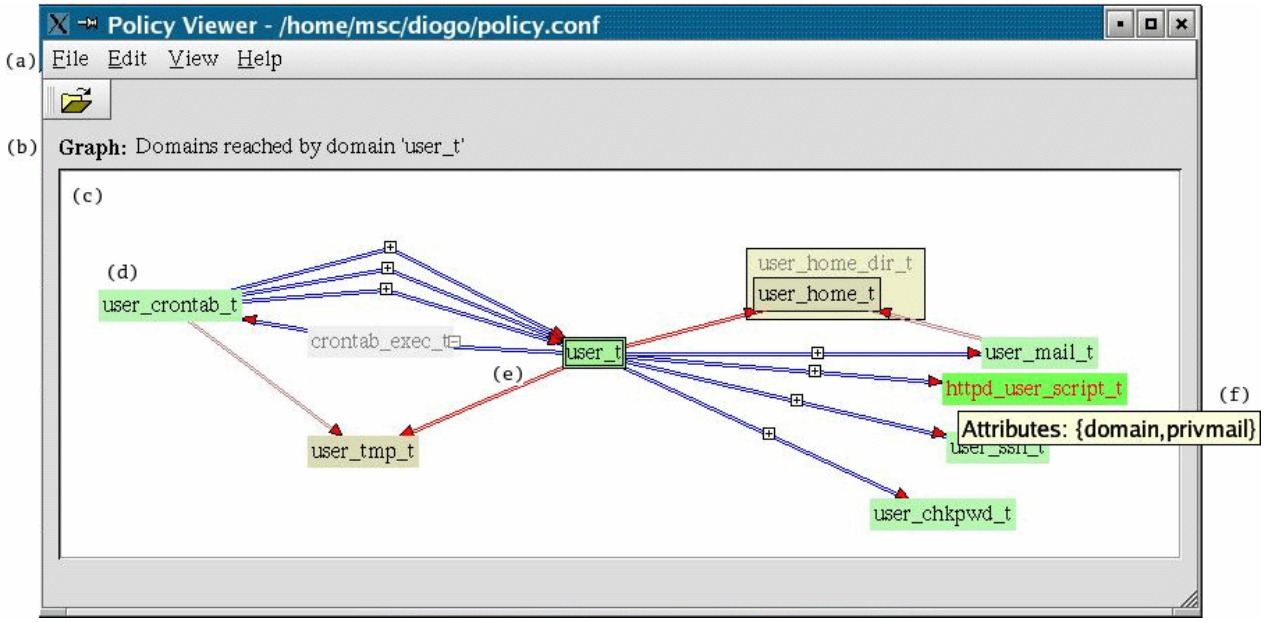


Figura 6.12: Interface básica com o usuário da ferramenta do **Policy Viewer**.

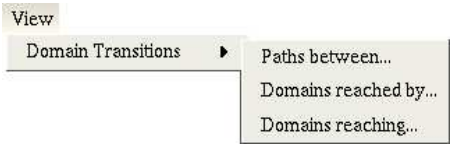


Figura 6.13: Opções do menu *View*

opção selecionada foi observar as transições de domínios possíveis a partir do domínio *user_t* (*Domains reached by...*, com alguns arestas de transição de tipo ocultas.

O *GraphDisplayWidget* é o coração da interface das representação visuais com o usuário. É ele a responsável por exibir o grafo para o usuário, e de direcionar os eventos gerados pelo usuário e pelo restante do sistema aos componentes de gerenciamento. Através dela, o usuário pode arrastar os nós, ajeitando-os conforme queira, requerer mais detalhes sobre um determinado nó ou aresta, ou exibir e esconder elementos da visualização. Adicionalmente, permite a inclusão de opções especificas aos diagramas utilizados. A Figura 6.14(a) apresenta o menu de contexto genérico com opções de ocultar ou exibir nós dos grafo, enquanto a Figura 6.14(b) apresenta as opções específicas às arestas *TETransitionEdge*.

Uma característica adicional, prevista para a versão final do **Policy Viewer**, prevê a exibição de dois *GraphDisplayWidget* na mesma janela, permitindo a visualização de dois diagramas em paralelo. Por exemplo, o usuário poderia visualizar a diagrama de objetos do sistema para identificar objetos expostos, visualizando o diagrama de domínios e tipos

(a) Menu de contexto padrão com opções sobre nós

(b) Menu de contexto específico de arestas de tipo *TETransitionEdge*

Figura 6.14: Menus de Contexto

ao lado para identificar os caminhos de exposição, conforme apresentado no exemplo do arquivo `/etc/passwd` na Seção 6.2.4.

Nós e arestas são representados conforme projetado na Seção 6.2. Na interface apresentada na Figura 6.12, apenas nós *TypeNode* e arestas *TETransitionEdge* estão presentes, pois foram os únicos implementados na ferramenta. Cores distintas são utilizadas para representar nós e arestas com significado diferente, sendo utilizado fundo verde para indicar domínios e fundo em tom laranja para indicar demais tipos do sistema. De forma semelhante, setas azuis indicam transição de domínios enquanto setas vermelhas indicam transição tipos. Apesar da classe possuir a opção de modificar as cores utilizadas na representação, visando permitir ao usuário configurar dinamicamente opções de destaque em função de outras características do nó, a ferramenta dispõe no momento somente de uma configuração estática para as colorações.

Como pode ser observado, ainda na Figura 6.12, pode haver mais de um tipo executável responsável pela transição de um domínio para outro (arestas do domínio *user_crontab_t* para *user_t*), de forma que utilizam-se arestas "curvadas" para permitir a representação de múltiplas arestas. Ainda na mesma figura, é possível observar que foram adicionados detalhes (as pequenas caixas com o caracteres '+' e '-'), que selecionam a exibição ou não do tipo da aplicação (ou objeto *container*) envolvido na transição, conforme previsto no diagrama de domínios e tipos. São exibidos dois destes nós "fantasmas": *crontab_exec_t* (aplicação) e *user_home_dir_t* (diretório).

O último detalhe da interface principal apresentada é o uso de *hints* para exibir detalhes a política. Parando o mouse sobre um nó ou aresta, será exibido um *hint* contendo as informações importantes sobre ele. Assim, ao parar o mouse sobre um nó *TypeNode*, são exibidos os atributos do nó. Se o usuário desejar mais informações, ele pode utilizar o menu de contexto para acessar a janela de detalhes do objeto selecionado. A Figura 6.15 exibe a janela de informações adicionais sobre o tipo *httpd_user_script_t*, contendo informações sobre o arquivo que define a regra, qual a definição utilizada, e eventuais comentários (a classe *PolicyConverter* utilizou o padrão adotado pela política do SELinux, que posiciona

o comentário imediatamente antes da regra, para obter essa informação).

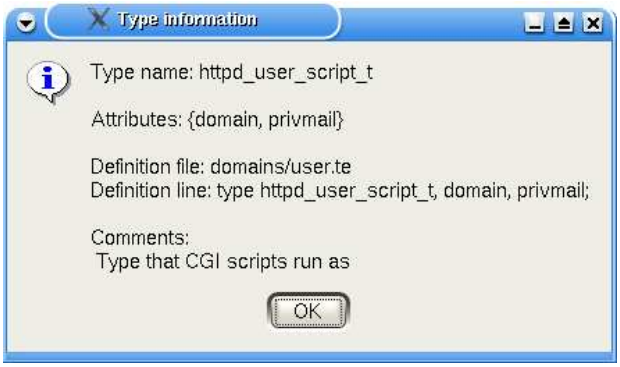
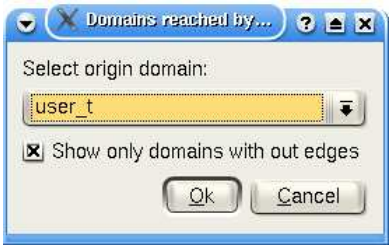
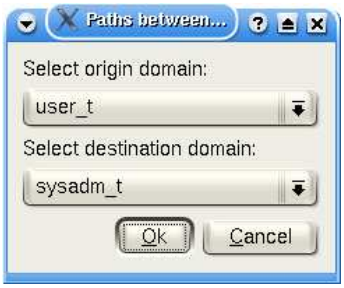


Figura 6.15: Janela exibindo detalhes de um nó

Além da janela principal, janelas auxiliares são utilizadas para seleção dos grafos a serem apresentados. A Figura 6.16 apresenta duas dessas janelas, acessadas a partir da opção *View* do menu principal. Em (a), é apresentada a janela de seleção do domínio de origem para exibição dos domínios acessíveis a partir do domínio selecionado (Figura 6.16(a)), e em (b) é apresentada a janela de seleção dos domínios de origem e destino de uma busca de caminhos de transição. Essas janelas cobrem os objetivos da implementação parcial da ferramenta, servindo de exemplo para as futuras janelas a serem desenvolvidas na implementação completa do projeto.



(a) Janela para seleção de nó para exibição da opção *Domain Transition: Reached by...*



(b) Janela para seleção de nó para exibição da opção *Domain Transition: Paths between...*

Figura 6.16: Janelas de seleção de visualização

6.4 Exemplos de Uso

Nessa seção, serão apresentados alguns exemplos de uso da ferramenta desenvolvida, demonstrando sua utilidade com relação aos objetivos a que se propõe: permitir melhor compreensão da política e auxiliar na identificação de riscos à segurança.

Para apresentar os exemplos, utilizou-se a ferramenta desenvolvida para gerar os grafos de transição de domínios e tipos. Como a ferramenta ainda não dispõe dos dados referentes as regras de acessibilidade, quando necessário foi utilizado o protótipo para gerar as informações adicionais. Serão apresentados 3 exemplos de uso, todos baseados na política padrão do SELinux, demonstrando o auxílio provido pela ferramenta na compreensão e verificação das políticas de segurança.

6.4.1 Identificação de caminhos entre domínios

Mesmo sem dispor da representação em grafo de todos os aspectos da política de segurança, o **Policy Viewer** permite realizar diversas análises na especificação. Suponha, por exemplo, que o administrador de segurança deseja saber quais são os domínios que um determinado usuário pode atingir. Utilizando a opção *Domains reached by...* do menu *View*, ele pode obter o diagrama de domínios e tipos a partir do domínio primário do usuário que é apresentado na Figura 6.17. A partir do grafo, é possível descobrir quais as aplicações envolvidas nas transições de domínio, e detalhes adicionais disponibilizados através do menu de contexto.

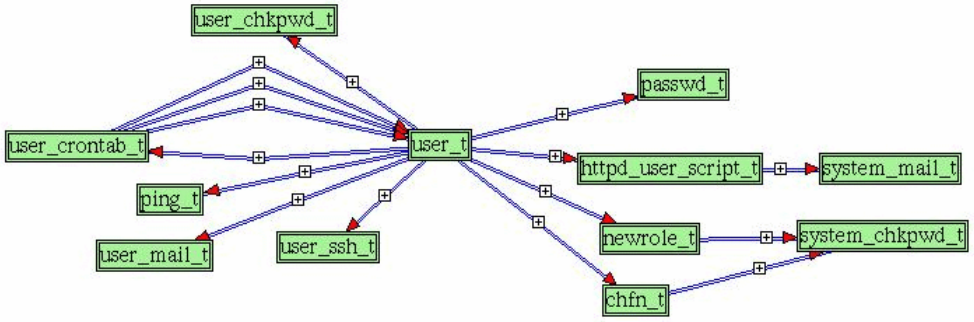


Figura 6.17: Domínios acessíveis a partir do domínio de usuário comum (*user_t*)

Outra informação de interesse está relacionada aos tipos dos novos arquivos que podem ser criados a partir de um determinado domínio. Dependendo de como foi configurada a política, esses tipos poder ser utilizadas para criar canais de comunicação entre aplicações em domínios isolados e ser utilizados para transmitirem informações não autorizadas. A Figura 6.18 apresenta o diagrama de transição de tipos para os domínios *user_t* e *sysadm_t*.

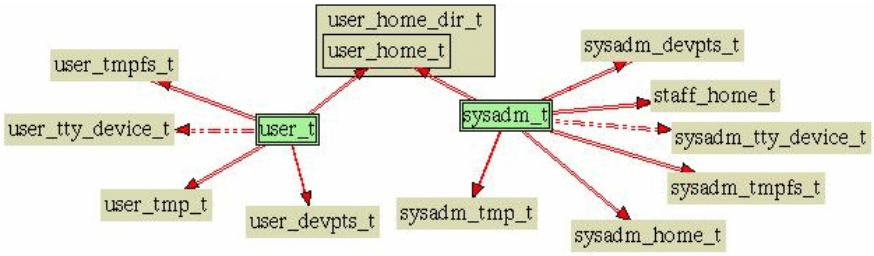
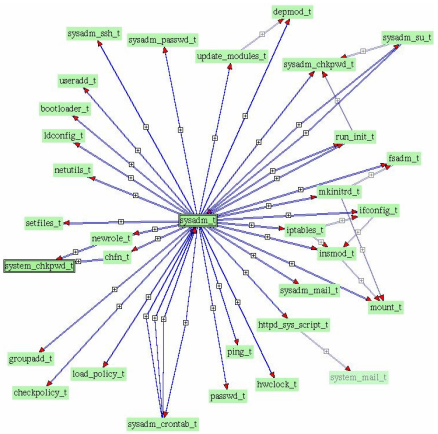
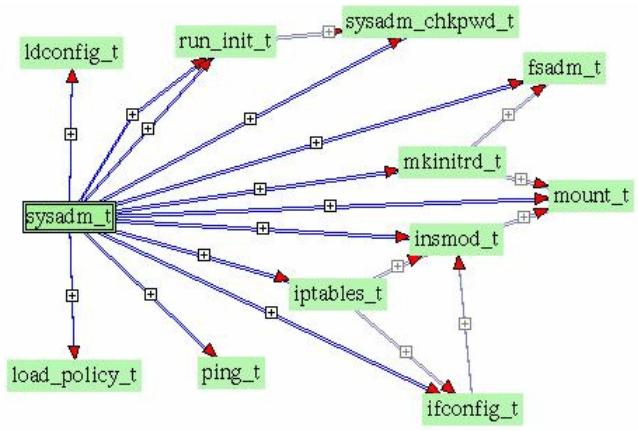


Figura 6.18: Domínios acessíveis a partir do domínio de usuário comum (*user_t*)

Eventualmente, a quantidade de nós obtidos em uma consulta é muito grande, por exemplo, devido ao número de transições de domínios sucessivas a partir de um domínio base ou devido ao número de regras de acessibilidade associadas, de forma que é necessário poder selecionar um subconjunto desses nós para permitir melhor visualização das regras. A Figura 6.19(a) apresenta o resultado obtido da consulta *Domains reached by...* a partir do domínio *sysadm_t*. Repare a grande quantidade de nós e arestas presentes, que podem causar certa confusão ao analisar a política. Utilizando recursos da ferramenta, é possível selecionar um conjunto de nós, e visualizá-los separadamente dos demais. A Figura 6.19(b) apresenta o mesmo diagrama, após a seleção de domínios ligados à administração de módulos, sistemas de arquivos, rede e políticas.



(a) Diagrama completo, com todos os domínios obtidos inicialmente.



(b) Diagrama após ocultar alguns domínios. Dessa forma, pode-se realizar uma análise mais detalhada de determinado aspecto da política.

Figura 6.19: Domínios acessíveis a partir do domínio de administração (*sysadm_t*

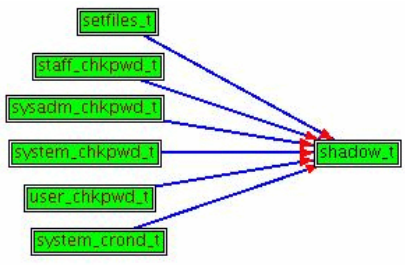
Mais do que simplesmente permitir visualizar a política, a ferramenta tem por objetivo auxiliar na identificação de eventuais falhas de segurança, que podem não ser capturadas

por verificadores de regras. Por isso, foram selecionados exemplos exibindo o uso da ferramenta com esse objetivo, que são apresentados a seguir.

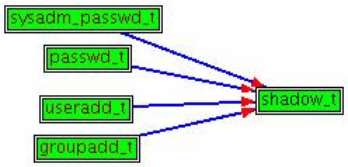
6.4.2 Exposição do arquivo de senhas

Um risco de segurança existente nos sistemas operacionais consiste na possibilidade de um atacante obter acesso de leitura ou escrita ao arquivo de senhas. Dessa forma, ele pode obter acesso legítimo ao sistema, contornando todo isolamento imposto pela política de segurança às aplicações vulneráveis, e dificultando a identificação do seu acesso não autorizado. No caso de permissão de leitura somente, várias técnicas já são utilizadas com sucesso para impedir que as senhas sejam descobertas, como é o caso das técnicas de cifragem. Entretanto, caso o atacante obtenha permissão de escrita, ele pode modificar a senha de qualquer usuário que deseje, ou ainda inserir um novo usuário no sistema. A política de segurança deve, portanto, impedir o acesso ao arquivo de senhas que não seja realizado através de aplicações específicas para esse fim.

Para verificar se a política padrão do SELinux trata corretamente essa situação utilizando o **Policy Viewer**, primeiramente identificou-se o tipo associado ao arquivo de senhas. Como a ferramenta, no estágio atual de desenvolvimento, ainda não dispõe do diagrama de objetos do sistema, utilizou-se o comando `ls -Z /etc/shadow`[2] para descobrir o tipo associado (no caso, *shadow_t*). A etapa seguinte é descobrir quais domínios tem acesso ao arquivo. A Figura 6.20 apresentam os domínios com permissão de leitura somente(a) e permissão de escrita (b) sobre objetos do tipo *shadow_t*.



(a) Domínios com acesso somente de leitura ao tipo *shadow_t*



(b) Domínios com acesso de escrita ao tipo *shadow_t*

Figura 6.20: Diagrama de domínios e tipos: regras de acessibilidade

Descobertos os domínios com acesso ao arquivo de senhas, decidiu-se por analisar aqueles que permitem escrever no arquivo de senhas, visto que o problema da leitura é

[2] A opção -Z foi adicionada ao comando ls do SELinux para retornar os contextos de seguranças associados aos arquivos listados

resolvido utilizando cifragem. Portanto, o objetivo agora é identificar quais os caminhos que transições sucessivas podem levar aos domínios *passwd_t*, *sysadm_passwd_t*, *useradd_t* e *groupadd_t*. Utilizando a implementação parcial do **Policy Viewer**, foi possível obter as transições que permitem atingir esses domínios, e quais aplicações permitem a transição. A Figura 6.21 apresenta o diagrama obtido. Repare que poucos domínios podem atingir os domínios que escrevem no arquivo de senhas e que, excluindo-se o caminho proveniente do domínio do administrador do sistema, o único domínio acessível é o *passwd_t*, e ainda assim, a transição só ocorre através de aplicações do tipo *passwd_exec_t*. Ao analisar as definições, obtém-se que a única aplicação associada a este tipo é a aplicação `/usr/bin/passwd`. A conclusão, portanto, é de que um ataque proveniente da rede não tem como ser bem sucedido na tentativa de corromper o arquivo de senhas, pois não há nenhum conjunto de transição de domínios que permita o acesso ao domínio *passwd_t* a partir de um domínio de aplicação de rede. Ainda assim, mesmo que ele logre acesso a um domínio autorizado de usuário (exceto, claro, o de administrador do sistema operacional), ele só poderá realizar tentar modificar a senha do próprio usuário, na ausência de vulnerabilidades na aplicação `/usr/bin/passwd`.

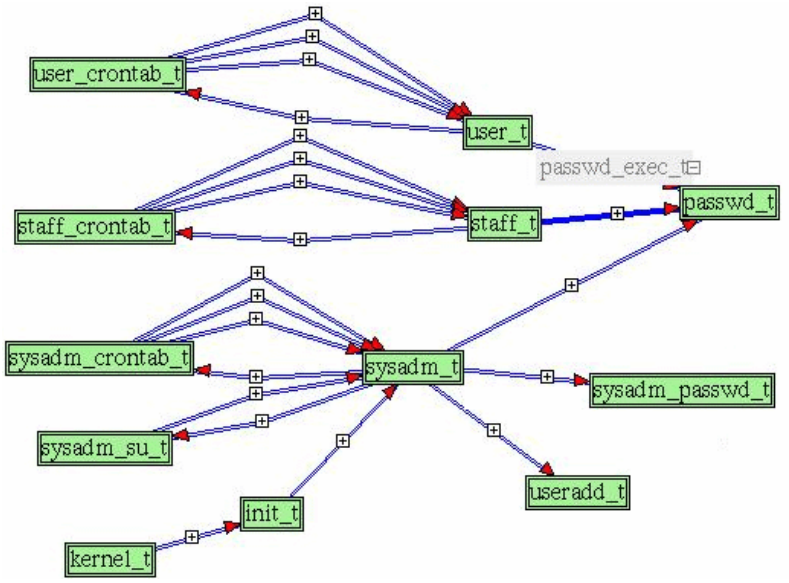


Figura 6.21: Diagrama de Domínios e Tipos. Domínios com acesso aos domínios *passwd_t*, *sysadm_passwd_t* e *useradd_t*

Para questão de comparação, a Figura 6.22 apresenta uma conjunto de comandos em `Shell Script` que poderiam ser utilizados para identificar, a partir da forma original da política (em definição textual), as relações apresentadas nas figuras anteriores. Primeiramente, identificam-se os domínios que têm permissão de escrita sobre o tipo *shadow_t*. A etapa seguinte é identificar quais os domínios que podem realizar transição para esse

domínio. Repare que, na definição dessa regra, ao invés de declarar domínio a domínio, utiliza-se um atributo comum a eles para designá-los, o que faz necessário mais uma consulta para determinar os domínios com esse atributo. A seguir, são identificados, recursivamente, os domínios que podem atingir o domínio *passwd_t* a partir do domínio *user_t*, o que torna esse trabalho muito oneroso para ser feito manualmente, e após vários comandos fica impossível interpretar os dados obtidos. Um detalhe adicional, a ser observado, é que a flexibilidade na especificação das regras podem esconder algumas regras, como aconteceria se, ao invés de querer descobrir quais domínios podem realizar transição para o domínio *passwd_t*, fosse desejado saber se existe uma transição do domínio de usuário comum (*user_t* para o domínio do *passwd_t*. Uma consulta direta utilizando esses dois domínios na consulta não retornaria nenhuma regra.

```
bash-2.05b$ cat policy.conf | egrep "^(audit)?allow.*shadow_t:.*write.*"
allow passwd_t shadow_t:file { create ioctl read getattr lock write...
allow sysadm_passwd_t shadow_t:file { create ioctl read getattr lock...
allow useradd_t shadow_t:file { create ioctl read getattr lock write...
allow groupadd_t shadow_t:file { create ioctl read getattr lock write...

bash-2.05b$ cat policy.conf | egrep "^type_.* passwd_t;$"
type_transition { userdomain  } passwd_exec_t:process passwd_t;

bash-2.05b$ cat policy.conf | egrep "^type .*userdomain.*"
type sysadm_t, domain, privlog, privowner, admin, userdomain,...
type staff_t, domain, userdomain, unpriv_userdomain,web_client_domain,...
type user_t, domain, userdomain, unpriv_userdomain, web_client_domain,...

bash-2.05b$ cat policy.conf | egrep "^type_.* user_t;$"
type_transition user_crontab_t { bin_t shell_exec_t }:process user_t;

bash-2.05b$ cat policy.conf | egrep "^type_.* user_crontab_t;$"
type_transition user_t crontab_exec_t:process user_crontab_t;

bash-2.05b$ ...
```

Figura 6.22: Identificando características da política de segurança utilizando sua forma textual.

6.4.3 Analisando a segurança do Apache

No exemplo anterior, explorou-se a possibilidade de invasão através do acesso ao arquivo de senhas, que permitiria ao atacante acesso "legítimo" ao sistema operacional. Neste

exemplo, o objetivo é verificar que, na eventualidade de comprometimento de um determinado serviço ou aplicação, um atacante não possa ir mais além do que o domínio necessário a aplicação. Para tal, faz-se necessário verificar as permissões associadas ao domínio da aplicação, e verificar eventuais transições de domínio que poderiam causar quebra da segurança do sistema.

Para exemplo, decidiu-se pela análise das políticas associadas a um Servidor HTTP (no caso, o Apache). A primeira etapa é identificar as transições de domínio possíveis a partir do domínio base (*httpd_t*), e analisar eventuais domínios de risco. A Figura 6.23 apresenta o diagrama de transição de domínios obtidos a partir do **Policy Viewer**, já destacando informações relevantes à segunda etapa da análise. Como pode ser observado, o compromentimento do Apache não apresenta riscos de acesso direto aos domínios de usuários do sistema, visto que as transições só permitem acesso a domínios de uso do Apache, exceto pelo acesso do domínio do servidor de e-mails (*system_mail_t*). Através dessa transição, o atacante pode utilizar a máquina invadida para enviar mensagens com origem falsa. Gerando o diagrama de acessibilidade do domínio *system_mail_t*, observa-se que ele só tem permissão de escrita em conexões de rede na porta 25 (servidores de e-mail) e a eventuais canais *fifo* criados a partir do domínio de origem, de forma que ele não poderá alterar o arquivo de configurações e estará restrito as opções configuradas pelo administrador do sistema para o servidor de e-mails.

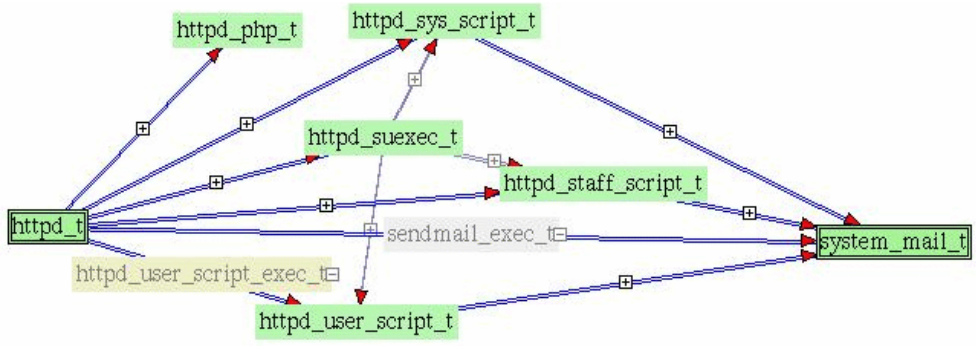
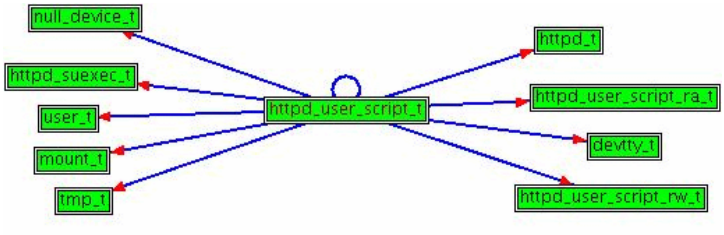


Figura 6.23: Domínios acessíveis a partir do domínio do servidor Apache (*httpd_t*)

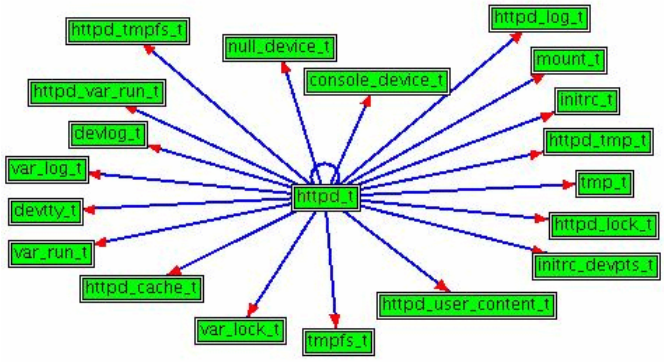
A necessidade ou não de restringir o acesso do Apache ao servidor de e-mails depende muito das objetivos de segurança da organização em que a política está implantada e das restrições que podem ser implementadas através das configurações do servidor de e-mail, de forma que a transição para o domínio *system_mail_t* pode ser considerada de risco ou não.

Ainda analisando o Apache, existe o risco de que o atacante utilize o acesso ao sistema de arquivos para "plantar" uma aplicação maliciosa no sistema, que pode ser executada inadivertidamente por um usuário, expondo seu domínio ao atacante. Para verificação,

foi escolhido analisar os domínios de usuários comuns que poderiam ser afetados por esse tipo de ataque. A Figura 6.24 apresenta as permissões de escrita dos domínios *httpd_user_script_t* (a) e *httpd_t* (b).



(a) Permissões de escrita do Domínio *httpd_user_script_t*



(b) Permissões de escrita do Domínio *httpd_t*

Figura 6.24: Diagrama de domínios e tipos: Permissões do Apache

Repare que o grande número de arestas dificulta identificar as arestas ligando os dois domínios aos tipos *user_t* e *httpd_user_content_t*. Para facilitar a visualização, utilizou-se o protótipo para gerar o diagrama (Figura 6.25) levando em conta apenas domínios que contivessem no nome '*httpd*' e '*user*'. Observe que, isolando os nós de interesse, fica muito mais simples identificar os riscos à segurança do sistema. No caso das arestas associadas associadas ao domínio *httpd_user_script_t*, a análise deos detalhes não indicam nenhum risco a segurança (a aresta de permissão de escrita ligada ao tipo *user_t* se refere a canais de comunicação *fifo* do sistema, e não a arquivos do usuário). No caso do domínio *httpd_t*, entretanto, a aresta representa uma permissão de escrita em arquivos do tipo *httpd_user_content_t* (associado a arquivos no diretório `public_html` do usuário).

Uma rápida procura em fóruns de discussão constatou que o problema já tinha sido levantado anteriormente, e que nas versões mais novas da política a permissão de escrita

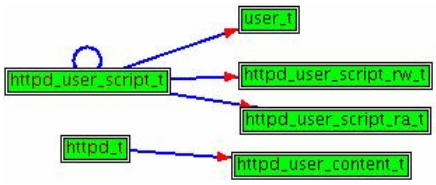


Figura 6.25: Regras de acessibilidade de domínios '*httpd*' para tipos '*user*'

foi retirada, uma vez que não há motivos para o Apache precisar escrever nada nesses diretórios, ficando a permissão para tal a cargo do domínio *http_user_script_t* e, ainda assim, em arquivos designados especificamente para esse fim (Tipos *httpd_user_script_rw_t* e Tipos *httpd_user_script_ra_t*).

6.5 Conclusão

Neste capítulo, foi apresentado o projeto da ferramenta **Policy Viewer**, os detalhes da implementação parcial da ferramenta e exemplos de uso da ferramenta, para validar suas funcionalidades. As modificações inseridas no projeto provenientes de constatações realizadas durante o desenvolvimento da ferramenta foram essenciais para o amadurecimento do projeto. Sua estrutura, dividida em componentes responsáveis por aspectos específicos, permite não só o desenvolvimento incremental do projeto, como também a adição de novas classes para expandir as funcionalidades da ferramenta. Apesar de ter sido feito especificamente para o sistema operacional SELinux, pequenas alterações na estrutura permitem o tratamento de outras políticas de segurança, bem como a inclusão de novas representações adequadas a finalidades não previstas ao longo do projeto.

A ferramenta desenvolvida até o momento, apesar de contar com apenas um subconjunto das funcionalidades previstas, permite realizar várias análises sobre a política, que auxiliam na identificação de três das setes questões de segurança previstas na fase de análise de requisitos. A fácil manipulação dos nós e arestas, a possibilidade de obter detalhes sobre demanda, e a representação da política independente do conhecimento da estrutura da especificação permitem ao administrador de políticas uma compreensão da política em vários níveis de abstração, dessa forma auxiliando-o na identificação de problemas dificilmente detectáveis através da análise das definições. Os exemplos de uso apresentado, mesmo utilizando somente a parte do projeto implementada, permitem validar esse objetivo e vislumbrar o potencial da ferramenta completa.

Capítulo 7

Conclusão

O surgimento e evolução da Internet trouxe inúmeros benefícios a organizações e usuários de computadores ao permitir o intercâmbio de informações de maneira fácil e ágil. Entretanto, apesar dessas conexões contribuírem com uma melhora significativa na eficiência e competitividade das organizações, elas também levaram ao aumento dos riscos de vazamento de informações sigilosas e invasões a sistemas computacionais, expondo computadores e seus sistemas a ataques provenientes do mundo todo.

Com isso em vista, foi iniciado um estudo focando a segurança de sistemas operacionais, identificado arquiteturas, mecanismos de segurança e características necessárias a um sistema operacional seguro. Constatou-se que muito já foi feito nesse sentido, incluindo o desenvolvimento de sistemas operacionais seguros. Entretanto, também constatou-se que esses sistemas operacionais estavam restritos aos meios acadêmicos, e que a grande maioria dos sistemas operacionais usados atualmente derivam sua estrutura de sistemas operacionais anteriores a Internet, e por isso não levam em conta inúmeras características de segurança necessárias ao contexto atual.

Várias são os motivos da fraca adoção de sistemas operacionais mais seguros, mesmo com o crescimento praticamente exponencial dos incidentes de segurança ano após ano, dentre os quais destaca-se o uso de políticas de segurança. Responsáveis por especificar o comportamento esperado do sistema operacional, as políticas de segurança criam abstrações dos elementos do sistema operacional para simplificar a definição das regras e reduzir o trabalho necessário para configurar todo o sistema. Porém, dada a quantidade de usuários, processos, arquivos e recursos existentes em um sistema operacional, a especificação torna-se muito extensa, o que dificulta a compreensão e a verificação da política definida. Foram estudados vários modelos, alguns com forte embasamento matemático, outros derivados das necessidade do mundo real, mas todos estavam sujeitos ao esse mesmo problema.

Esses estudos culminaram com o desenvolvimento do projeto de uma ferramenta para

visualização gráfica de políticas de seguranças, denominada **Policy Viewer**, baseado no sistema operacional SELinux, objetivo principal desta dissertação. Todas as etapas realizadas, desde os estudos preliminares, até a análise de requisitos e o estudo de trabalhos relacionados foram apresentadas, detalhando as características necessárias identificadas, a arquitetura do projeto, as estruturas de dados utilizadas e as interfaces desenvolvidas.

Para validar a ferramenta com relação ao seu objetivo inicial, a de auxiliar na compreensão e verificação das políticas de segurança, e assim contribuir na disseminação do uso de sistemas operacionais seguros, foi realizada a implementação parcial da ferramenta a partir de um subconjunto das funcionalidades previstas. Como os exemplos demonstram, mesmo possuindo apenas parte das requisitos propostos, a implementação parcial da ferramenta desempenha muito bem o seu papel de representar as políticas de segurança de forma visual, mantendo uma representação clara e independente do conhecimento da sintaxe de especificação da política, quebrando a linearidade imposta pela definição textual e auxiliando na análise da política especificada.

Uma característica adicional obtida devido a forma como foi estruturado o projeto é a possibilidade de estender a ferramenta para representar a política de outros sistemas operacionais, ou ainda de utilizar outras formas de representação, sem ser necessário reestruturar totalmente a aplicação. Isolando as funcionalidades em componentes específicos, e fazendo a interligação entre os componentes através de um componente mediador, é possível adicionar funcionalidades ao sistema modificando apenas o componente associado a funcionalidade.

7.1 Trabalhos Futuros

Existem várias possibilidades de extensões ao projeto proposto nesta dissertação. A primeira, e mais importante, já foi citada ao longo do texto, e consiste em adaptar a ferramenta para outros sistemas operacionais e políticas de segurança. Apesar de não ter sido proposto um *framework* para uso genérico, a estruturação componentizada da ferramenta permite uma adaptação mais fácil, através da modificação do componente envolvido na conversão da política.

Outro ponto a ser estudado é a possibilidade de utilizar outras representações que não a estrutura de grafos. Grafos foram utilizados por haver uma grande disponibilidade de mecanismos de manuseio, e por possuir propriedades semânticas similares as das regras da política. Entretanto, outras formas de representação poderiam ser empregadas para apresentar características distintas da política, focando determinados aspectos não presentes em grafos. Estudos envolvendo a área de visualização de informações poderiam contribuir na identificação de representações visuais aplicáveis, que poderiam ser anexadas a ferramenta **Policy Viewer** para prover informações de forma mais clara e selecionada para o

usuário.

Ainda há trabalho a fazer no componente Gerenciador de Exibição, uma vez que foi praticamente negligenciado durante o projeto e prototipação. No momento, apenas um mecanismo de disposição dos nós do grafo está previsto (*Force Directed Graph*), porém existe uma vasta gama de mecanismos que poderiam ser estudados e adaptados, permitindo ao usuário da ferramenta selecionar mecanismos diferentes para destacar aspectos distintos da política.

Finalmente, um trabalho mais complexo que pode ser derivado da ferramenta é o desenvolvimento de um editor completo de políticas de segurança, totalmente integrado e dispondo de mecanismos de visualização, verificação e distribuição de políticas de segurança em sistemas operacionais conectados em rede. Um editor nesses moldes simplificaria muito o processo de gerenciamento de política, dessa forma estimulando ainda mais a adoção de sistemas operacionais seguros.

Apêndice A

Glossário

ACL *Access Control List* - Lista de Controle de Acesso

AVC *Access Vector Cache* - *Cache* do vetor de permissões de acessos

API *Application Programming Interface* - Interface de programação da aplicação

DAC *Discretionary Access Control* - Controle de Acesso Discricionário

DTE *Domain and Type Enforcement* - Imposição de Domínio e Tipo

GID *Group Identifier* - Identificador de Grupo associado a arquivos do Linux

IBAC *Identity-Based Access Control* - Controle de Acesso baseado em identidade

IPC *Inter-proccess Call* - Chamada entre processos

LDAP *Lightweight Directory Access Protocol*

LIDS *Linux Intrusion Detection System*

MAC *Mandatory Access Control* - Controle de Acesso Mandatório

RBAC *Role-Based Access Control* - Controle de Acesso baseado em papéis

RSBAC *Rule Set Based Access Control* - Controle de acesso baseado em conjuntos de regras

SELinux *Security-Enhanced Linux*

SID *Security Identified* - Identificador de Seguranca do SELinux

TCSEC *Trusted Computer System Evaluation Criteria* - Critério de avaliação de sistemas computacionais confiáveis

TE *Type Enforcement* - Imposição de Tipo

UID *User Identifier* - Identificador de Usuário associado a arquivos do Linux

UML *Unified Modeling Language*

Referências Bibliográficas

[1] D. Baker. Fortresses built upon sand. In *Proceedings of the New Security Paradigms Workshop*, 1996.

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

[3] D. E. Bell and L. J. La Padula. Secure computer system: Unified exposition and multics interpretation. Technical Report MTR-2997 Rev. 1, MITRE Corporation, Bedford, MA, EUA, March 1975.

[4] Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin G. Sirer, Marc Fiuczynski, David Becker, Susan Eggers, and Craig Chambers. Extensibility, safety and performance in the SPIN operating system. *Proceeding of the 15th Symposium on Operating System Principles*, 1994.

[5] K. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, EUA, April 1977.

[6] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 4th edition, November 2003.

[7] David F. C. Brewer and Michael J. Nash. The chinese wall security policy. *Proceedings of The IEEE Symposium on Research in Secutity and Privacy*, pages 206–214, May 1989.

[8] Stuart K. Card, Jock D. Mackinlay, and Ben Shneierman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1999.

[9] D. Clark and D. Wilson. A comparison of commercial and military security policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 184–194, April 1987.

[10] C. Cowan, Calton Pu, Dave Maier, Heather Hinton, Jonathan Walpole andPeat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Conference*, 1998.

[11] Celmar Guimarães da Silva and Heloisa Vieira da Rocha. Contribuições de visualização de informação para a área de educação a distância. In *Anais do VI Simpósio sobre Fatores Humanos em Sistemas Computacionais (IHC 2004)*, Outubro 2004.

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

[13] DoD. *Trusted Computer Security Evaluation Criteria*. Department of Defense, 1985. DOD 5200.28-STD.

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

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

[16] Halldor Fossa and Morris Sloman. Interactive configuration management for distributed object systems. In *EDOC '97: Proceedings of the 1st International Conference on Enterprise Distributed Object Computing*, pages 118–128. IEEE Computer Society, 1997.

[17] Simson Garfinkel and Gene Spafford. *Pratical Unix & Internet Security*. O'Reilly & Associates, Inc., USA, 2nd edition, 1996. 971 p.

[18] Alain Gefflaut, Trent Jaeger, Yoonho Park, Jochen Liedtke, Kevin Elphinstone, Volkmar Uhlig, Jonathon Tidswell, Luke Deller, and Lars Reuther. The sawmill multiserver approach. *9th SIGOPS European Workshop*, September 2000.

[19] D. Gifford. Cryptographic sealing for information secrecy and authentication. *Communications of the ACM*, 25(4):274–286, April 1982.

[20] Hermann Hartig, Oliver Kowalski, and Winfried Kuhnhauser. The birlix security architecture. *Gesellschaft fur Mathematik und Datenverarbeitung*, June 92.

[21] Trent Jaeger, Xiaolan Zhang, and Fidel Cacheda. Policy management using access control spaces. *ACM Trans. Inf. Syst. Secur.*, 6(3):327–364, 2003.

[22] Manuel Koch, Luigi V. Mancini, and Francesco Parisi-Presicce. A graph-based formalism for rbac. *ACM Trans. Inf. Syst. Secur.*, 5(3):332–365, 2002.

[23] Diogo Ditzel Kropiwiec and Paulo Lício de Geus. Paradigmas de segurança em sistemas operacionais. In *Anais do IV Workshop de Segurança em Sistemas Computacionais*, Gramado/RS, Maio 2004.

[24] Demetrios Lambrou. Tcpa enabled open source platforms, september 2003. Disponível em <`http://www.crazylinux.net/downloads/projects/TCPA/TCPA\_thesis.html`>. Acesso em: 27/02/04.

[25] B. Lampson. Protection. In *Proccedings of the fifth Princeton Symposium of Information Science and Systems*, pages 437–443, March 1971.

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

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

[28] P. Loscocco and S. Smalley. Meeting critical security objectives with security-enhanced linux. In *Proccedings of The 2001 Ottawa Linux Symposium*, Ottawa, Ont., Canada, July 2001.

[29] P. A. Loscocco, S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrel. 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, October 1998.

[30] G.K. Mostefaoui and P. Brezillon. Modeling context-based security policies with contextual graphs. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 28–32, march 2004.

[31] Sape J. Mullender, Guido van Rossum, Andrew S. Tanenbaum, Robbert van Renesse, and Hans van Staveren. Amoeba: A distributed operating system for the 1990s. *Computer*, 23(5):44–53, 1990.

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

[33] R. M. Needham and R. D.H. Walker. The cambridge cap computer and its protection system. In *SOSP '77: Proceedings of the sixth ACM symposium on Operating systems principles*, pages 1–10. ACM Press, 1977.

[34] Matunda Nyanchama and Sylvia Osborn. The role graph model and conflict of interest. *ACM Trans. Inf. Syst. Secur.*, 2(1):3–33, 1999.

[35] Amon Ott. Rule set based access control (rsbac). In *the Snow Unix Event - unix.nl congress "Reliable Internet"*, Waardenburg, September 2001.

[36] Birgit Pfitzmann, James Riordan, Christian Stüble, Michael Waidner, and Arnd Weber. The PERSEUS system architeture. Technical report, IBM Research Division, September 2001.

[37] W. Polk. Approximating Clark-Wilson access triples with basic unix controls. In *Proceedings of the 4th USENIX UNIX Security Symposium*, pages 145–154, October 1993.

[38] Dennis M. Ritchie and Ken Thompson. The UNIX time-sharing system. *Commun. ACM*, 17(7):365–375, 1974.

[39] Luciana Alvim Santos Romani and Heloisa Vieira da Rocha. Intermap: Visualizando a interação em ambientes de educação a distância baseados na web. In *Anais do I ENCUENTRO INTERNACIONAL DE INFORMÁTICA EN LA EDUCACIÓN SUPERIOR - INFOUNI ' 2001*, Habana, Cuba, 2001.

[40] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 47 – 63, Berlim, Germany, 2000. ACM Press, New York, NY, USA.

[41] J. S. Shapiro, S. J. Muir, J. M. Smith, and D. J. Farber. EROS: A capability system. Technical Report MS-CIS-97-03, Distributed Systems Laboratory - University of Pennsylvania, Philadelphia, PA, USA, June 97.

[42] S. D. Smalley. Configuring the SELinux policy. Technical report, National Security Agency of United States of America, January 2003.

[43] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask Security Architecture: System support for diverse security policies. *Proceedings of the Eighth USENIX Security Symposium*, pages 123–139, August 1999.

[44] Andrew S. Tanenbaum and Albert S. WoodHull. *Operating Systems: Design and Implementation*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2nd edition edition, 1997.

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

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