



FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Tecnologia em Segurança da Informação

Gustavo William de Oliveira Pontes

Controle de acesso mandatório em sistemas Linux
com o módulo SELinux

Americana, SP

2017



FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

Gustavo William de Oliveira Pontes

Controle de acesso mandatório em sistemas Linux

com o módulo SELinux

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação, sob a orientação do Professor Rossano Pablo Pinto.

Americana, SP

2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

P858c PONTES, Gustavo William de Oliveira

Controle de acesso mandatório em sistemas Linux: com o módulo SELinux. / Gustavo William de Oliveira Pontes. – Americana, 2017.

61f.

Monografia (Curso de Tecnologia em Segurança da Informação) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Rossano Pablo Pinto

1 Segurança em sistemas de informação 2. Linux – sistema operacional I.
PINTO, Rossano Pablo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.518.5

681.3.066


Gustavo William de Oliveira Pontes

Controle de acesso mandat3rio em sistemas Linux
com o m3dulo SELinux

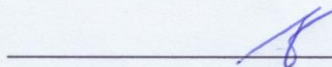
Trabalho de Conclus3o de Curso desenvolvido em cumprimento 3 a exig3ncia curricular do Curso Superior de Tecnologia em Seguran3a da Informa3o, sob a orienta3o do Professor Rossano Pablo Pinto.

Americana, 11 de Dezembro de 2017.

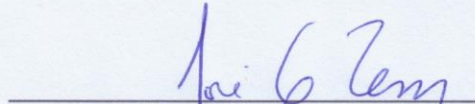
Banca Examinadora:



Rossano Pablo Pinto (Presidente)
Professor Mestre
Faculdade de Tecnologia de Americana



Clerivaldo Jos3 Roccia (Membro)
Professor Mestre
Faculdade de Tecnologia de Americana



Jos3 Luis Zem (Membro)
Professor Doutor
Faculdade de Tecnologia de Americana

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus por toda a força, iluminação e apoio que tem conferido a mim para que fosse possível chegar até aqui.

Aos meus pais por todo amor, carinho, apoio e incentivo dado, pontos importantíssimos para a realização deste feito

Dentre meus amigos, dedico os mais profundos sentimentos de agradecimento à Eliana Carvalho, por todo o incentivo, conselhos e positividade transmitida durante todo o período de elaboração deste estudo e ao meu amigo Danilo Miranda, por manter-se presente a todo momento, estando à disposição para ajudar-me com o que fosse necessário.

Aos meus chefes, que dispuseram de boa vontade e compreensão, apoiando-me e sendo flexíveis em relação à realização de tarefas e horários de trabalho.

A todos os meus demais amigos, que sempre forneceram todo apoio necessário e incentivos.

Aos meus professores, pelo apoio e boa vontade de repassar seus conhecimentos durante as aulas e fora deste período.

A minha professora Maria Cristina Aranda, por diversos momentos onde necessitei de sua ajuda e orientação e assim pude contar com ela, sempre disposta e pronta a me ajudar, transmitindo imenso carinho, que é recíproco.

Ao meu professor orientador Rossano Pablo Pinto, quem dispôs de seu precioso tempo durante o expediente e fora deste período para orientar-me, tornando assim possível a realização deste trabalho. Agradeço-o profundamente também por ser responsável pelo meu alicionamento nesta área, cuja qual pretendo seguir carreira.

RESUMO

Ciberataques geram prejuízo anual de quase meio bilhão de dólares para a economia mundial e a preocupação com Segurança da Informação cresce a cada dia. Os sistemas operacionais representam importante parcela na segurança de dados. Este trabalho visa alcançar o objetivo de demonstrar a eficácia de se utilizar o modelo de controle de acesso mandatário nos sistemas operacionais para endurecer a segurança. Endurecer a segurança é importante porque devido à existência de falhas e limitações do modelo de controle de acesso discricionário presente nos sistemas operacionais, hackers podem explorar diversas vulnerabilidades dos softwares computacionais para manipular de maneira ilegítima dados contidos em toda a Internet. A fim de endurecer a segurança de um sistema operacional Linux com controle de acesso mandatário, aplicam-se sobre a API LSM módulos, como o SELinux, que permitem restringir de maneira muito mais precisa e eficaz qualquer interação que um indivíduo possa vir a ter com um sistema, tornando-o assim mais seguro.

Palavras Chave: Controle de acesso mandatário. Linux. LSM. Segurança da Informação. SELinux

ABSTRACT

Cyberattacks cause annual losses of about half a billion dollars to the world economy and the information security concern grows every day. The operating systems represent an important part of data security. This document objective is to show the effectiveness of using mandatory access control model to security hardening. Security hardening is so important due the existence of issues and limitations in the discretionary access control model present in the operating systems, hackers can exploit various vulnerabilities to illegitimately manipulate data across all the Internet. To harden the operating system security, modules, such SELinux, are applied to the LSM API, which allow to more accurate and effectively restrict any interaction between a person and a system, turning the system more safe from hackers.

Keywords: Mandatory access control. Linux. LSM. Information Security. SELinux.

SUMÁRIO

| | |
|---|-----------|
| 1 INTRODUÇÃO | 1 |
| 1.1 Objetivo geral | 1 |
| 1.2 Objetivos específicos | 1 |
| 1.3 Justificativa | 2 |
| 1.4 Delimitação do escopo | 2 |
| 1.5 Estrutura do trabalho | 2 |
| 2 SEGURANÇA DA INFORMAÇÃO | 3 |
| 2.1 Controle de acesso | 4 |
| 2.2 Segurança da informação nas organizações | 6 |
| 3 SISTEMAS OPERACIONAIS | 9 |
| 3.1 Controle de acesso em sistemas operacionais | 10 |
| 3.2 Processos e entidades do sistema | 11 |
| 3.3 Controle de Acesso Discricionário..... | 12 |
| 3.3.1 Manipulação de privilégios | 17 |
| 3.4 <i>Capabilities</i> | 23 |
| 3.5 Controle Acesso Mandatório | 25 |
| 4 LINUX SECURITY MODULES | 27 |
| 4.1 Abstração fornecida pela LSM..... | 28 |
| 4.2 Exemplos de módulos LSM | 29 |
| 5 SELINUX | 31 |
| 5.1 Maneiras de restrição com SELinux | 32 |

| | |
|---|----|
| 5.2 Conceitos técnicos..... | 34 |
| 5.2.1 Rótulos | 34 |
| 5.2.2 Type Enforcement - Regras | 38 |
| 6 CENÁRIOS | 39 |
| 6.1 Cenário 1: Controlando a execução de um programa | 39 |
| 6.2 Cenário 2: Manipulação de permissões | 41 |
| 6.3 Cenário 3: Eficácia do SELinux..... | 46 |
| 6.4 Discussão dos resultados | 49 |
| 7 CONSIDERAÇÕES FINAIS | 50 |
| REFERÊNCIAS..... | 51 |
| APÊNDICE A - CONFIGURAÇÃO, FUNCIONAMENTO E SINTAXE DO SUDO..... | 54 |
| APÊNDICE B - SINTAXE DO COMANDO "SU" | 57 |
| APÊNDICE C - PROCESSO DE ATRIBUIÇÃO E VISUALIZAÇÃO DO SUID..... | 58 |
| APÊNDICE D - ESTRUTURA DE UM MÓDULO SELINUX | 59 |

1 INTRODUÇÃO

A grande explosão tecnológica dos últimos anos permitiu a substituição em nível mundial do armazenamento de dados em papel por infraestruturas computacionais capazes de armazenar dados em quantidades muito maiores demandando menor espaço físico e utilização de recursos naturais. Além disso, o compartilhamento de informações entre indivíduos em locais diferentes foi facilitado com a capacidade de compartilhamento de informações via Internet. Com isso, antes a preocupação em proteger estas informações deixou agora de ser orientada apenas a proteger fisicamente o meio que as armazena, se estendendo também ao ambiente lógico criado pelos sistemas computacionais.

Embora existam mecanismos nativos dos sistemas operacionais para aplicar controles de segurança sobre os dados, esses mecanismos possuem algumas limitações e vulnerabilidades que são descobertas e exploradas por *hackers* em ritmo maior do que são corrigidas.

Surge assim a necessidade da aplicação de mecanismos mais elaborados que permitam restrições maiores de acesso para trabalhar conjuntamente com os mecanismos nativos.

1.1 Objetivo geral

Explicitar como a aplicação de controle de acesso mandatório sobre um sistema operacional é capaz de reduzir efeitos provocados por vulnerabilidades em programas ou até mesmo programas mal intencionados.

1.2 Objetivos específicos

- Apresentar princípios gerais de Segurança da Informação;
- Apresentar a API (LSM) responsável pela disponibilidade de utilização de controle de acesso mandatório em sistemas Linux;
- Apresentar o SELinux como mecanismo de controle de acesso mandatório e demonstrar sua eficácia;

1.3 Justificativa

Este trabalho justifica-se no fato do número de incidentes de Segurança da Informação ter superado 2,4 milhões de ocorrências somente no Brasil entre 2014 e 2016 (CERT.BR, 2017) e crimes cometidos pela Internet gerarem perda anual de US\$ 445 bilhões à economia mundial (O GLOBO, 2015). Desta forma torna-se plausível que se utilize o máximo possível de recursos capazes de endurecer a segurança eletrônica. A utilização de controle de acesso mandatório é capaz de endurecer a segurança em elevados níveis, tanto que é frequentemente utilizado em instalações militares (ROUSE, 2013).

1.4 Delimitação do escopo

Existem diversos métodos e extensões ao modelo DAC (Controle de Acesso Discricionário) que implementam a segurança no sistema operacional, no entanto estes métodos não serão abordados, mas quando o forem, serão de maneira superficial, apenas para conceituar elementos importantes que levam a motivação para se utilizar o modelo MAC (Controle de Acesso Mandatório). Dentro do escopo da abordagem do modelo MAC, haverá breve explanação conceitual da API LSM (Linux Security Módulos), para que se possa entender como as permissões MAC são tratadas pelo sistema. Além disso, em termos de módulos LSM, apenas o módulo SELinux será abordado. A abordagem do SELinux será apenas conceitual, com destaque em sua política *Targeted*. Esta abordagem visa demonstrar apenas processos básicos pertinentes ao gerenciamento do módulo, bem como expor a eficácia do modelo MAC em relação ao modelo DAC.

1.5 Estrutura do trabalho

Este trabalho tem seu conteúdo em seis capítulos. O primeiro trata conceitos fundamentais sobre segurança da informação. O segundo capítulo conceitua sistemas operacionais, seus modelos de controle de acesso e vulnerabilidades. O terceiro capítulo apresenta informações sobre a API LSM. O quarto capítulo explana a utilização do SELinux como módulo de controle de acesso mandatório em sistemas operacionais Linux. O quinto capítulo exemplifica através de três cenários a utilização do SELinux, mostrando sua eficácia. Finalmente o sexto capítulo apresenta considerações finais.

2 SEGURANÇA DA INFORMAÇÃO

Segurança é o que torna bens invulneráveis aos riscos que estes podem vir a sofrer. É indispensável para qualquer fim, seja ele pessoal ou profissional, já que indivíduos e organizações necessitam manter seus bens seguros.

Segundo Kolling (2010) esses dados tornam-se informações quando passam a ter valor para indivíduos e ou organizações.

Cassa (2007) afirma que na grande maioria dos casos, negócios de natureza pública ou privada são sustentados pela informática e que a era atual é a da tecnologia da informação, na qual a informação torna-se um bem com valor mensurável, acordando assim com o raciocínio de Kolling (2010).

Baseando-se na afirmação de Cassa (2007) então pode-se associar a evolução da tecnologia da informação nos últimos anos com o fato da sociedade e seus negócios passarem a ser dependentes dela para aumentar sua produtividade. Assim todas as informações antes armazenadas em papel são agora blocos de dados eletrônicos armazenados em sistemas computacionais, sistemas estes que possuem vulnerabilidades de segurança devido à características técnicas.

Considerando então que a existência de vulnerabilidades no meio onde as informações residem aliada ao valor agregado a estas, conferem ao comprometimento da segurança o poder de causar impacto negativo aos proprietários das informações.

Para reduzir o risco de serem impactados, indivíduos e organizações necessitam que suas informações sejam protegidas. Surge assim a preocupação com a Segurança da Informação.

Conclui-se então que a SI (Segurança da Informação) é um conceito que ajuda manter informações livres de acidentes e incidentes que possam causar prejuízo, sejam estes de qualquer espécie. A SI baseia-se em três pilares que juntos são conhecidos como CID (Confidencialidade, Integridade e Disponibilidade).

Kolling (2010) salienta que Segurança da Informação aborda a proteção de determinados dados com o intuito de preservar seus respectivos valores para organizações ou indivíduos.

Conclui-se também que a função da SI, é então em sua forma mais básica, aplicar controles a fim de garantir os já citados pilares CID e também – entre outros - os de autenticidade e irretratabilidade como princípios a serem seguidos, provendo autenticidade às informações. Assim há a garantia de que todo o seu conteúdo é proveniente de seu criador ou parte autorizada a modificá-lo. Também há garantia que os indivíduos ou organizações que exerceram ação sobre uma informação não possam repudiar suas ações, sejam elas ações de má-fé, ou de natureza não intencional.

Considerando então a sensibilidade das informações e seus valores para seus proprietários, do ponto de vista das organizações a Segurança da Informação tem de ser preocupação vital para garantir o funcionamento do negócio, visto que as informações são o principal pilar de qualquer negócio sustentável.

Recapitulando, todas os conceitos apresentados anteriormente conferem à informação a dignidade de ser classificada como um bem. Este bem, possui relação direta com outros bens, portanto sua segurança retorna à premissa descrita no primeiro parágrafo: Segurança é indispensável para qualquer fim, seja ele pessoal ou profissional já que indivíduos e organizações necessitam manter seus bens seguros.

Para isso, é preciso a aplicação de métodos que originam-se no conceito de controle de acesso, tema abordado na seção 2.1.

2.1 Controle de acesso

O instinto protetivo é intrínseco dos seres do reino animalia. Atos instintivos como a proteção territorial e do alimento, são exemplos natos da aplicação de meios de controle de acesso, ainda que primitivos.

O conceito de controle de acesso prevê qualquer forma de proteção sobre um bem, sendo ele um objeto, recurso ou serviço, independentemente da existência no meio real ou virtual.

Então, controle de acesso é primordial para a existência da segurança e que é com a aplicação de mecanismos de controle de acesso que é possível manter a segurança de bens, protegendo-os de subtrações, exposições, danos provenientes de ações não autorizadas e qualquer outra violação acometida por parte não autorizada.

A importância da aplicação de controle de acesso sobre um bem abrange os mesmos pontos da necessidade de manter sua segurança, sendo o impacto às partes envolvidas com este bem proporcional ao valor agregado a ele.

De acordo com Costa (2012), o termo controle de acesso é considerado uma referência ao ato de conceder acesso a um local ou recurso apenas para pessoas autorizadas a acessá-lo, através de meios de controle que podem ser recursos humanos, como um porteiro, ou recursos mecânicos como fechaduras, segredos ou recursos eletrônicos. No caso de recursos virtuais ainda existem os mecanismos lógicos por meio de autenticação, autorização e auditoria.

Como a SI (Segurança da Informação) abrange tanto a existência da informação no meio físico que à armazena quanto no meio virtual que a trata, fazem parte dos métodos de controle: criação de senhas complexas, implementação de *firewall's*, entre outros.

Além disso, é necessário também que seja realizado o controle de acesso físico na organização. Uma sala que contém data centers ou arquivos impressos de caráter sigiloso, por exemplo, são locais que não podem ter passe livre para todos os colaboradores. As informações e dados sigilosos da empresa só devem ser acessados por usuários privilegiados, cada um com sua respectiva senha de acesso, para evitar o vazamento e roubo de informações confidenciais (SANTOS, 2016).

Seguindo o raciocínio, fica clara a necessidade de proteção não apenas no nível lógico, mas também no físico, onde arquivos e documentos que possuam informações de caráter confidencial cujo vazamento possa causar grande impacto na organização, como por exemplo, folhas de pagamento podem vir a causar situações indesejáveis entre funcionários caso divulgadas, ou informações sobre projetos ainda não lançados que caso venham a conhecimento da concorrência resultem em prejuízo à organização, entre outras informações que devem ter sua confidencialidade preservada.

Fato pertinente ao assunto é que quanto maior o grau de proteção para acessar determinado objeto, mais complexo e menos prático o acesso a ele se torna e em alguns casos devido à aplicação de mecanismos de controle em demasia, ainda que o valor e

necessidade de proteção do objeto sejam elevados, cresce o risco de negligenciar a segurança, abrindo-se precedentes a um possível incidente de segurança.

2.2 Segurança da informação nas organizações

Dada a importância da SI e do controle de acesso, empresas tem adotado cada vez mais medidas e controles buscando evitar perdas. Segundo o O Globo, (2015) o gasto com SI no país cresce em ritmo anual entre 30% e 40%.

Com isso, muitas empresas instituem em seu organograma, profissionais especializados na área, quase sempre formando um comitê de segurança da informação.

Normalmente o comitê de segurança da informação tem em sua composição além de profissionais da área, funcionários de todas as áreas da empresa, a fim de alinhar todos os departamentos do negócio com os requisitos de SI.

Utilizando como exemplo o Escritório de Governança de TI do Tribunal Regional da 11ª Região (2011), responsável pelos estados de Amazonas e Roraima, é papel do comitê de SI:

- Estabelecer diretrizes sobre as iniciativas de SI perante toda a organização;
- Prestar suporte às iniciativas estabelecidas;
- Alinhar os objetivos institucionais e a tecnologia da informação com a SI;
- Apoiar atividades pertinentes à SI na gestão de riscos no que se diz a avaliação, aceitação e tratamento de riscos;
- Acompanhar e propor planos de ação para melhor aplicar a política de SI da organização;

Política de SI (PSI), segundo Oliveira (2013), é um documento composto por um conjunto de métodos, normas e procedimentos, que devem ser amplamente divulgados dentro da organização, além de estar em constante revisão.

Visto o papel do comitê, é importante ressaltar que este age em conformidade com a política de Segurança da Informação da instituição, bem como também é função do

comitê periodicamente revisar minimamente cada detalhe desta política, alterando-a sempre que houver a necessidade.

Além disso, uma PSI necessita ser clara, concisa, de fácil compreensão e traz como benefícios o comprometimento do alto escalão com a continuidade dos negócios e o aumento da conscientização por parte de todas as áreas da empresa quanto a SI (BR DATA SECURITY, 2004).

Uma PSI, para impor suas normas, métodos e procedimentos, muitas vezes prevê a implementação de mecanismos de controle de acesso. É de grande valia ressaltar que este requisito não se limita a proteger exclusivamente o meio físico ou o meio virtual. Segundo Zaniquelli (2010) é necessário que haja uma aliança de mecanismos de controle de acesso físicos e lógicos para criar um sistema de proteção com maior abrangência, capaz de proteger o meio físico que contém a informação e também proteger o meio lógico que à gerencia e disponibiliza.

Seguindo o raciocínio, em uma organização, a fim de proteger dados armazenados em uma determinada máquina, é necessário mais do que instalar fechaduras biométricas nas portas de acesso ao local que ela se encontra, ou alocar um profissional da área de segurança patrimonial para inibir acessos não autorizados. Passa a ser necessário também que se utilize de meios de controle de acesso lógicos para garantir a segurança destes dados, visto que a informação contida nesta máquina - graças à evolução da TI (Tecnologia da informação) nos últimos anos - fica disponível a outros indivíduos utilizando outras máquinas, podendo estar em locais remotos a este que está sendo protegido fisicamente.

Além de adotar todas essas medidas internamente, é importante que as organizações também às testem a fim de garantir a efetividade das medidas tomadas em barrar tentativas de acesso indevidas, conforme dito por Rocha (2013). Para isto, contratam empresas terceiras especializadas em *pentests* (serviço que permite colocar à prova cada um dos pontos da segurança da organização, sejam eles lógicos ou físicos).

Essa prática, segundo a empresa Profissão Hacker (2017), é chamada de *Ethical Hacking* e vem conquistando a cada ano mais destaque. Além de testar a segurança, o *Ethical Hacking* fornece à organização contratante relatórios completos de auditoria e

aponta nestes relatórios todas as vulnerabilidades encontradas - dentro de um escopo pré acordado entre contratante e contratada – e possíveis correções que podem ser aplicadas à elas.

Com isso, as organizações podem garantir níveis maiores de segurança, já que grande parte das vulnerabilidades que não puderam ser encontradas ou resolvidas pelos profissionais de SI internos são apontadas pelos relatórios fornecidos por estas empresas terceiras.

Um dos principais tópicos mencionados nesses relatórios é a segurança dos sistemas operacionais. Estes representam importante parte nas camadas de *software* existentes em um computador. Portanto o capítulo 3 aborda conceitos de controle de acesso em sistemas operacionais.

3 SISTEMAS OPERACIONAIS

Graças à evolução da TI a preocupação com as informações de uma organização deve ser estendida ao meio lógico das máquinas onde estão armazenadas.

Máquinas necessitam de um conjunto de procedimentos ordenados para realizar sua função. Para ser possível oferecer suas funções aos seus usuários, esses dispositivos precisam ser gerenciados por programas de computador que exercem justamente a função de executar as instruções automaticamente.

Segundo Koprowiec e Geus (2004, p. 1) essa função fornece a abstração entre *hardware* e aplicativos através do gerenciamento dos recursos físicos da máquina, otimizando-os de forma que seja possível seu uso para efetuar cálculos e executar instruções.

Os programas que exercem esta função são os chamados sistemas operacionais, cuja função é fornecer abstração entre o *hardware* e os aplicativos, permitindo através destes a interação humana com estes dispositivos.

Com isso, é viabilizado um meio de armazenar informações nestes dispositivos de forma similar à organização dos dados em papel, em arquivos e pastas. Assim é possível disponibilizar estas informações aos usuários em um ambiente virtual amigável e intuitivo que é exposto pelos aplicativos.

Além de fornecer abstração entre *hardware* e aplicativos e viabilizar o armazenamento de dados, os sistemas operacionais também oferecem a possibilidade de compartilhar informações entre dois ou mais sistemas distintos via rede, desta forma permitindo a criação de um enorme compartilhamento entre dispositivos, tal como ocorreu com a grande rede mundial de computadores conhecida como Internet.

Estando então compartilhados estes dados entre dispositivos e usuários em uma rede mundial, surge a preocupação com a segurança destes dados e com o controle de acesso a eles.

Assim, tendo em vista a proporção tomada pelo compartilhamento de informações na Internet, a segurança dos sistemas operacionais que suportam o compartilhamento é

imprescindível, uma vez que estas informações armazenadas nos sistemas computacionais e compartilhadas via rede possuem tanto valor e sensibilidade quanto em outrora, quando eram armazenadas em papel e compartilhadas por meios físicos e presenciais.

Além de imprescindível a preocupação em segurança, considerando apenas a igualdade do valor e sensibilidade dessas informações, estando elas armazenadas em papéis ou em sistemas computacionais, esta preocupação deve ser intensificada uma vez que o compartilhamento na internet sem devidos cuidados com controle de acesso lógico aumenta consideravelmente o risco de exposição e comprometimento das informações já que controlar o acesso físico ao meio que estão armazenadas não bastará para protegê-las e nem sempre incidentes que possam vir a acontecer serão detectados, ou ainda que detectados é possível que a detecção seja tardia ou que não seja possível impedir a sua ocorrência à tempo.

3.1 Controle de acesso em sistemas operacionais

Vulnerabilidades em sistemas operacionais são descobertas todos os dias em ritmo maior do que é possível corrigi-las. Koprowiec e Geus (2004, p. 2) esclarecem que: “A segurança desses sistemas operacionais reside nos direitos de acesso de um usuário (ou grupo de usuários) a um recurso (arquivo, periférico, memória, etc.) [...]”.

Como o sistema operacional é o meio primário de interação entre o ser humano e a máquina, muitas vulnerabilidades de *softwares* que estão sendo executados em uma máquina podem ser contornadas com o endurecimento da segurança do sistema operacional, o que pode ser feito controlando os direitos de acesso dos usuário no sistema, seguindo o raciocínio dos autores Koprowiec e Geus (2004, p.2).

A fim de controlar os direitos de acesso dos usuários no sistema, existem dois modelos principais utilizados nos sistemas Linux. Apesar de tratar os direitos de acesso de forma distinta, a aliança destes dois modelos pode oferecer endurecimento efetivo do sistema contra vulnerabilidades de *softwares* instalados sobre ele.

São estes modelos:

[...] DAC - Controle de Acesso Discricionário, e MAC - Controle de Acesso Mandatório. No primeiro, a segurança dos arquivos fica à discrição do usuário que o possui, e estes podem alterar as permissões de seus arquivos com relação a si próprio e a outros de forma livre. No segundo, existem políticas de segurança, comuns ou específicas a cada usuário, que restringem as ações que este pode realizar sobre o arquivo, visando não só a segurança do usuário, mas também a segurança de todo o sistema operacional. (KOPROWIEC; GEUS, 2004, p. 4).

3.2 Processos e entidades do sistema

No plano real, em termos de controle de acesso, pode-se autorizar uma pessoa em particular ou um grupo de pessoas a ter acesso a um objeto. No plano lógico, em um sistema operacional, estas entidades são tratadas como “usuários” e “grupos”. Cada usuário possui um identificador numérico único denominado “*User ID*” (UID). O mesmo ocorre com grupos, que tem seu identificador denominado “*Group ID*” (GID). Em um sistema Linux, o usuário “root” e o grupo “root”, por padrão sempre terão o identificador único igual a zero.

Além destas, há outra entidade denominada “processos”. Os processos, assim como qualquer entidade no meio lógico possuem um identificador único chamado “*Process ID*” (PID). Qualquer ação que os usuários realizem sobre o sistema é feita através de programas, que se tornam processos no momento em que o usuário os executa.

É através dos identificadores numéricos destas entidades, que aplica-se controle de acesso em sistemas operacionais.

Um processo é uma abstração utilizada pelo Linux para representar um programa em execução. É o objeto por meio de qual o uso de memória, tempo de processador e recursos de E/S de um programa podem ser gerenciados e monitorados (NEMETH et al., 2012, p. 38).

Assim, qualquer *software* (programa) em execução na máquina será representado, controlado e gerenciado através de um processo.

Como os *softwares* trabalham acessando e modificando recursos do sistema - sejam dispositivos, arquivos, outros processos, regiões de memória, conexões de rede, entre outros recursos - estas ações são concedidas ou negadas de acordo com as permissões do usuário que executou seu binário executável.

Para que isso seja possível, cada processo, além de seu PID, possui alguns atributos. Segundo Nemeth et al. (2012, p. 39), dentre estes atributos, estão:

- UID: Identificador do usuário que executou o binário do programa.
- EUID (*Effective User ID*): Identificador do usuário cujo qual detém as permissões que o processo irá ter sobre o sistema operacional.

Normalmente, os atributos “UID” e “EUID” irão referenciar o mesmo usuário. Porém, há casos em que programas necessitam de privilégios mais elevados do que o usuário que o executou possui para realizar algumas ações.

Assim, conforme descrito em 3.3.1 há a possibilidade de tornar o “EUID” diferente do “UID”, dando ao processo permissões para realizar estas ações.

Nemeth et al. (2012, p. 39) esclarece que a importância de manter os dois atributos é fazer com que a obtenção de privilégios baseada na alteração do “EUID” possa ser temporária, memorizando no atributo “UID” o identificador real do usuário que o executou. Assim, o “EUID” poderá assumir este valor quando não forem mais necessários os privilégios de outro usuário.

Além disso, os desenvolvedores podem definir regras que controlem o comportamento do programa de acordo com o “UID”, podendo restringir algumas ações ainda que o usuário referenciado no “EUID” tenha privilégios para realizá-las.

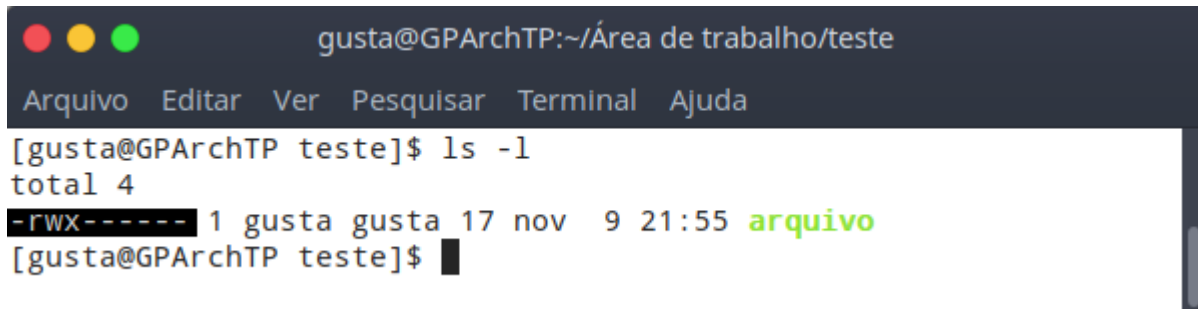
3.3 Controle de Acesso Discricionário

Controle de Acesso Discricionário ou DAC é o modelo de controle de acesso utilizado na maioria dos sistemas operacionais. Ele é baseado em descrever quais os privilégios que os usuários têm sobre cada recurso do sistema operacional.

O conceito inicial de DAC refere-se às permissões no sistema de arquivos, tomando-as como exemplo, serão abordadas inicialmente três permissões: leitura, escrita e execução. Embora este conceito permita restrição de privilégios a todos os usuários não afetados pelas permissões, para o usuário root de um sistema Linux, o DAC tradicional não é capaz de restringir qualquer permissão devido a forma como os processos são tratados pelo Linux Kernel – abordagem em 3.4.

Para visualizar as permissões de arquivos em um diretório, utiliza-se o comando “ls -l”, conforme é possível ver na Figura 1:

Figura 1 – Visualização de permissões com comando “ls -l”.



```
gusta@GPArchTP:~/Área de trabalho/teste
Arquivo Editar Ver Pesquisar Terminal Ajuda
[gusta@GPArchTP teste]$ ls -l
total 4
-rwx----- 1 gusta gusta 17 nov 9 21:55 arquivo
[gusta@GPArchTP teste]$
```

Fonte: O próprio autor.

Observe que o retorno do comando “ls -l” traz os seguintes campos sobre o arquivo “arquivo”, que está contido na pasta “teste”:

- Tipo: “-”;
- Permissões (Usuário/Grupo/Todos): “rwx-----”;
- Nº de Hard Links: “1”;
- Proprietário: “gusta”;
- Grupo: “gusta”;
- Data de modificação (AA/Mês/D/HH/MM): “17 nov 9 21:55”;
- Nome: “arquivo”;

Buscando entender rapidamente o funcionamento das permissões DAC, os atributos a serem destacados são “permissões”, “proprietário”, “grupo” e “arquivo”.

O atributo “tipo” contém apenas um caractere referente ao tipo de recurso e pode assumir os valores: “-” para arquivo, “b” para arquivo de blocos, “c” para arquivo de caracteres, “d” para diretório, “p” para *pipe* (mecanismo de comunicação entre processos baseado em sistema de arquivos) e “s” para *sockets*, entre outros tipos e valores.

O atributo “permissões” possui 9 caracteres e refere-se a três grupos de usuários do sistema, para facilitar o entendimento, este atributo é dividido em 3 blocos:

- O primeiro bloco refere-se às permissões que o usuário mostrado no atributo “proprietário” terá sobre o arquivo. Este bloco contém três caracteres, sendo que cada um representa um tipo de permissão. O primeiro caractere pode assumir tanto o valor “-” que significa que o usuário não terá permissão para ler o arquivo, quanto “r” que significa a concessão da permissão de leitura; O segundo caractere refere-se à permissão de escrita e pode assumir o valor “-” para impedir o usuário de escrever no arquivo ou “w” para permitir escrita no arquivo; O terceiro caractere, assim como os outros pode assumir o valor “-” para negar permissão de execução, ou “x” para conceder esta permissão.
- O segundo bloco possui também três caracteres e exerce a mesma função que o primeiro, salvo que ao invés de referir-se as permissões do proprietário do arquivo, refere-se a todos os usuários pertencentes ao grupo que é mostrado no atributo “grupo”.
- O terceiro bloco possui a mesma quantidade de caracteres que os dois anteriores e refere-se às permissões de todos os usuários, exceto o proprietário do arquivo ,e os usuários pertencentes ao grupo mostrado no atributo “grupo”.

O atributo “proprietário” tem a finalidade de identificar o usuário do sistema que é o criador ou dono do objeto.

O atributo “grupo” indica o grupo de usuários qual o arquivo pertence, geralmente este atributo é preenchido com o grupo de mesmo nome que o proprietário ou outro grupo que o proprietário pertença, porém pode ser alterado para qualquer grupo.

O atributo “nome” indica o arquivo cujo qual todos os outros atributos referem-se.

A fim de exemplificar o funcionamento do DAC , a Figura 2 ilustra algumas ações de diferentes usuários do sistema sobre um mesmo arquivo.

```

postgres@GPArchTP:/home/gusta/Área de trabalho/teste
Arquivo Editar Ver Pesquisar Terminal Ajuda
[gusta@GPArchTP teste]$ ls -l
total 4
-rwx----- 1 gusta gusta 17 nov  9 22:20 arquivo
[gusta@GPArchTP teste]$ cat arquivo
arquivo de teste
[gusta@GPArchTP teste]$ echo 'gusta' > arquivo
[gusta@GPArchTP teste]$ cat arquivo
gusta
[gusta@GPArchTP teste]$ su postgres
Senha:
[postgres@GPArchTP teste]$ cat arquivo
cat: arquivo: Permissão negada
[postgres@GPArchTP teste]$ echo 'postgres' >> arquivo
bash: arquivo: Permissão negada
[postgres@GPArchTP teste]$ su root
Senha:
[root@GPArchTP teste]# cat arquivo
gusta
[root@GPArchTP teste]# echo 'root' > arquivo
[root@GPArchTP teste]# cat arquivo
root
[root@GPArchTP teste]# chmod 704 arquivo
[root@GPArchTP teste]# exit
exit
[postgres@GPArchTP teste]$ ls -l
total 4
-rwx---r-- 1 gusta gusta 5 nov  9 22:22 arquivo
[postgres@GPArchTP teste]$ cat arquivo
root
[postgres@GPArchTP teste]$ echo 'postgres' > arquivo
bash: arquivo: Permissão negada
[postgres@GPArchTP teste]$ cat arquivo
root
[postgres@GPArchTP teste]$ su root
Senha:
[root@GPArchTP teste]# chmod 706 arquivo
[root@GPArchTP teste]# exit
exit
[postgres@GPArchTP teste]$ ls -l
total 4
-rwx---rw- 1 gusta gusta 5 nov  9 22:22 arquivo
[postgres@GPArchTP teste]$ cat arquivo
root
[postgres@GPArchTP teste]$ echo 'postgres' > arquivo
[postgres@GPArchTP teste]$ cat arquivo
postgres
[postgres@GPArchTP teste]$ █

```

Figura 2 – Exemplo de funcionamento das permissões DAC.

Fonte: O próprio autor.

É possível ver no primeiro comando (`ls -l`) que trata-se de um arquivo com permissões de leitura e escrita e execução apenas ao proprietário `gusta`. Nos próximos três comandos é possível notar que o usuário `gusta` não só é capaz de visualizar (`cat`) o conteúdo do arquivo, mas também de alterar (`echo`) seu conteúdo.

Quando o usuário `postgres` entra em cena, como este não tem permissões sobre o arquivo, o mesmo não pode sequer visualizar o conteúdo, tampouco alterá-lo, resultando qualquer uma destas tentativas em um aviso de permissão negada.

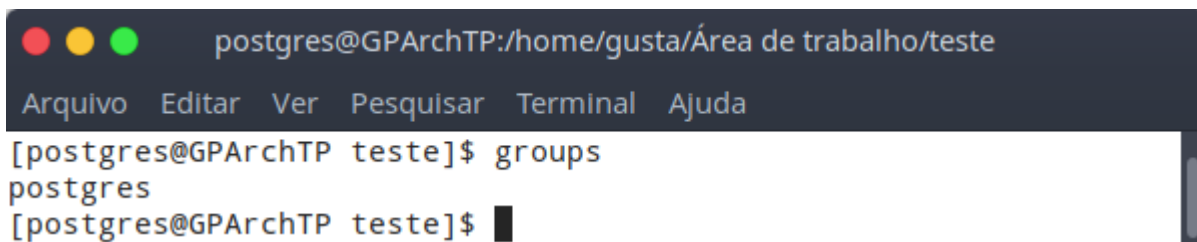
Então o usuário `root` que assim como o usuário `postgres` não possui nenhuma permissão sobre o arquivo, é capaz de visualizar e alterar o conteúdo do arquivo, porém isto ocorre porque, conforme explicado anteriormente, o usuário `root` possui controle total sobre qualquer recurso do sistema e, devido a isso, o DAC não é capaz de restringir seu acesso a qualquer que seja o recurso, ainda que exista a tentativa de negar acesso ao `root`.

Antes de o usuário `postgres` entrar em cena novamente, é possível observar que o usuário `root` alterou as permissões do arquivo (`chmod`) permitindo a todos os usuários verem o conteúdo, porém a permissão de alterar fica apenas ao proprietário.

Então, quando o usuário `postgres` tenta visualizar o conteúdo, tem sucesso, porém ao tentar alterar o conteúdo, o aviso de permissão negada é exibido e o conteúdo permanece o mesmo.

O usuário `root` então entra em cena e altera as permissões novamente, desta vez permitindo a todos usuários alterar o conteúdo. Assim quando o usuário `postgres` retorna a ativa, é capaz tanto de visualizar o conteúdo, quanto de alterá-lo.

É importante ressaltar que ao executar o comando `groups` para checar em quais grupos o usuário `postgres` está, o mesmo não encontra-se no grupo `gusta`, conforme é possível ver na Figura 3:

Figura 3 – Comando “groups”.

```
postgres@GPArchTP:/home/gusta/Área de trabalho/teste
Arquivo Editar Ver Pesquisar Terminal Ajuda
[postgres@GPArchTP teste]$ groups
postgres
[postgres@GPArchTP teste]$ █
```

Fonte: O próprio autor.

3.3.1 Manipulação de privilégios

Existem situações onde os usuários necessitam executar tarefas que requerem mais privilégios do que os que eles possuem. Para sanar esta necessidade, existem meios de permitir ao usuário obter estes privilégios.

Sudo (su "do") allows a system administrator to give certain users (or groups of users) the ability to run some (or all) commands as root while logging all commands and arguments (MILLER, [s.d.]).

O *Sudo*, como dito por Miller ([s.d.]), é um programa que permite usuários executarem comandos e executarem outros programas com privilégios de outro usuário (incluindo o usuário root) - sem possuir sua senha.

No entanto, é necessário que um usuário privilegiado defina quais outros usuários podem desfrutar das funcionalidades do *Sudo* através da edição do arquivo de configurações “/etc/sudoers”. Além disso, é feito registro em um arquivo de log sobre qualquer ação realizada enquanto o *Sudo* é utilizado. Também é importante ressaltar que o *Sudo* não permite a utilização de comandos assim como um interpretador *shell*, ou seja, alguns comandos como o “cd” para navegar diretórios e outros comandos afins não irão funcionar.

Para obter-se mais informações sobre configuração, funcionamento e sintaxe do *Sudo*, pode-se consultar o apêndice A.

Outro método de executar programas em nome de outro usuário – obtendo-se assim privilégios do outro usuário - para para executar comandos, é utilizar o comando “su”. Segundo The Linux man-pages project (2014), este comando permite instanciar um *shell* com privilégios de outro usuário.

Diferentemente do Sudo, como neste comando um *shell* é instanciado, torna-se possível navegar entre diretórios e suas ações não são registradas. No entanto para utilizar este comando é necessário conhecer a senha do usuário cujo qual deseja-se instanciar o *shell* com os privilégios, exceto se o utilizador do comando for o usuário root, que é capaz de instanciar o *shell* de qualquer usuário do sistema sem necessitar da senha do usuário.

É possível ver na Figura 4 que, de acordo com as permissões de cada diretório pessoal (ls -l), apenas o dono é capaz de acessá-los (ls). Quando o usuário gusta utiliza o comando “su”, é solicitada senha de root e então o *shell* passa a executar com privilégios de root, quando o usuário root utiliza o comando “su teste”, o *shell* passa a ser executado com privilégios de teste sem ter sido solicitada senha do usuário teste.

Quando o usuário teste utiliza o comando “su gusta”, antes de o *shell* passar a executar com privilégios de gusta, é solicitada a senha do usuário gusta.

Quando o usuário gusta utiliza novamente o comando “su”, também é solicitada a senha de root novamente, porém quando o usuário root utiliza o comando “su gusta”, o *shell* de gusta é instanciado sem que seja solicitada senha alguma, assim como aconteceu anteriormente quando o usuário root instanciou o *shell* do usuário teste.

Mais informações sobre a sintaxe do comando “su” podem ser obtidas através da leitura do apêndice B.

Para que se torne possível a manipulação de privilégios com os programas “sudo” e “su”, conforme descrito anteriormente, é necessário que os binários deles possuam o atributo SUID definido. Segundo Anne (2011), o SUID (*Set owner User ID up on execution*) é um atributo que permite obter os privilégios do usuário proprietário do programa. Em outras palavras, ao executar um programa quando este atributo está definido sobre seu binário executável, ao invés do processo ser executado com permissões do usuário que o está executando, os privilégios utilizados serão os do usuário proprietário do arquivo executável do programa, pois o EUID deste processo será igual ao UID do proprietário, ao invés do UID do usuário que o executou.

Figura 4 – Exemplo de utilização do comando “su”.

```

gusta@GPArchTP:~
Arquivo Editar Ver Pesquisar Terminal Ajuda
[gusta@GPArchTP ~]$ ls -l / | grep 01:18
drwxr-x--- 14 root root 4096 nov 12 01:18 root
[gusta@GPArchTP ~]$ ls -l /home
total 24
drwx----- 32 gusta gusta 4096 nov 12 01:18 gusta
drwx-----  2 root root 16384 ago 26 22:59 lost+found
drwx-----  7 teste teste 4096 nov 12 01:11 teste
[gusta@GPArchTP ~]$ ls /home/teste/
ls: não foi possível abrir o diretório '/home/teste/': Permissão negada
[gusta@GPArchTP ~]$ ls /root
ls: não foi possível abrir o diretório '/root': Permissão negada
[gusta@GPArchTP ~]$ ls /home/gusta
'Área de trabalho'  Modelos          Público
Documentos         Música           q
Downloads          NetBeansProjects 'systemctl disable teclado'
go                 pgadmin.log     Vídeos
Imagens            Projects        'VirtualBox VMS'
[gusta@GPArchTP ~]$ su
Senha:
[root@GPArchTP gusta]# ls /root
'Área de trabalho'  Downloads  Modelos  Público
Documentos         Imagens   Música   Vídeos
[root@GPArchTP gusta]# ls /home/teste
Desktop Documents Music Pictures Videos
[root@GPArchTP gusta]# ls /home/gusta
'Área de trabalho'  Modelos          Público
Documentos         Música           q
Downloads          NetBeansProjects 'systemctl disable teclado'
go                 pgadmin.log     Vídeos
Imagens            Projects        'VirtualBox VMS'
[root@GPArchTP gusta]# su teste
[teste@GPArchTP gusta]$ ls /root
ls: não foi possível abrir o diretório '/root': Permissão negada
[teste@GPArchTP gusta]$ ls /home/gusta
ls: não foi possível abrir o diretório '/home/gusta': Permissão negada
[teste@GPArchTP gusta]$ ls /home/teste
Desktop Documents Music Pictures Videos
[teste@GPArchTP gusta]$ su gusta
Senha:
[gusta@GPArchTP ~]$ su
Senha:
[root@GPArchTP gusta]# su gusta
[gusta@GPArchTP ~]$ █

```

Fonte: O próprio autor.

Para exemplificar o funcionamento do SUID, o programa apresentado na Figura 5 foi utilizado. A função de tal programa consiste basicamente em abrir um arquivo de texto chamado “confidencial” e contido na pasta onde o binário executável do programa estiver salvo, ler cada um de seus caracteres e exibi-los em tela. Caso haja problemas com a leitura do arquivo, então o programa exibe a mensagem “Não foi possível exibir o arquivo, verifique se você tem permissão para isso”.

Figura 5 – Programa em C para demonstração do SUID.

```
int main(int argc, char** argv) {
    FILE *arquivo;
    char caractere, caminho[] = "confidencial";
    arquivo = fopen(caminho, "r");
    if (arquivo != NULL){
        while ((caractere=fgetc(arquivo)) != EOF) {
            printf("%c", caractere);
        }
    }
    else {
        printf("Não foi possível exibir o arquivo, verifique se você tem permissão para isso.\n");
    }
    return 0;
}
```

Fonte: O próprio autor.

A Figura 6 retrata testes com o programa da Figura 5 a fim de demonstrar como o sistema trata permissões de leitura e execução quando o SUID está definido. O primeiro fato a se considerar na Figura 6 é que, de acordo com as permissões (ls -l) do arquivo “confidencial”, ele apenas pode ser visualizado e modificado pelo root. Quanto ao arquivo “programa”, trata-se do binário executável do programa demonstrado na Figura 5 e pode ser executado por qualquer usuário do sistema (rwxr-wr-w).

Como é possível ver, o usuário *gusta* é incapaz de ler (cat) o conteúdo do arquivo “confidencial”, portanto o programa “programa” rodando com privilégios do usuário *gusta* é incapaz de exibir em tela o conteúdo do arquivo, exibindo a mensagem de falha.

Porém, quando o usuário *root*, que é proprietário do arquivo define o SUID para o binário do programa (chmod u+s programa), automaticamente estabelece que qualquer usuário com permissões para executar o programa o fará com os mesmos privilégios de usuário *root* para qualquer ação que o programa realizar. Assim, quando o usuário *gusta* executa-o, consegue visualizar o conteúdo do arquivo “confidencial”.

Figura 6 – Demonstração do SUID.

```

gusta@GPArchTP:~/Área de trabalho/teste
Arquivo  Editar  Ver  Pesquisar  Terminal  Abas  Ajuda
gusta@GPArchTP:~/Área de t... x  gusta@GPArchTP:~/Área de t... x  +
[gusta@GPArchTP teste]$ ls -l
total 16
-rw----- 1 root root  17 nov 13 00:21 confidencial
-rwxr-xr-x 1 root root 8560 nov 12 11:53 programa
[gusta@GPArchTP teste]$ cat confidencial
cat: confidencial: Permissão negada
[gusta@GPArchTP teste]$ /home/gusta/Área\ de\ trabalho/teste/programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[gusta@GPArchTP teste]$ exit
exit
[root@GPArchTP teste]# chmod u+s programa
[root@GPArchTP teste]# ls -l
total 16
-rw----- 1 root root  17 nov 13 00:21 confidencial
-rwsr-xr-x 1 root root 8560 nov 12 11:53 programa
[root@GPArchTP teste]# su gusta
[gusta@GPArchTP teste]$ cat confidencial
cat: confidencial: Permissão negada
[gusta@GPArchTP teste]$ /home/gusta/Área\ de\ trabalho/teste/programa
Conteudo Secreto
[gusta@GPArchTP teste]$ exit
exit
[root@GPArchTP teste]# echo 'Alterado por root' >> confidencial
[root@GPArchTP teste]# cat confidencial
Conteudo Secreto
Alterado por root
[root@GPArchTP teste]# su gusta
[gusta@GPArchTP teste]$ cat confidencial
cat: confidencial: Permissão negada
[gusta@GPArchTP teste]$ /home/gusta/Área\ de\ trabalho/teste/programa
Conteudo Secreto
Alterado por root
[gusta@GPArchTP teste]$ exit
exit
[root@GPArchTP teste]# chmod u-s programa
[root@GPArchTP teste]# su gusta
[gusta@GPArchTP teste]$ ls -l
total 16
-rw----- 1 root root  35 nov 13 00:38 confidencial
-rwxr-xr-x 1 root root 8560 nov 12 11:53 programa
[gusta@GPArchTP teste]$ /home/gusta/Área\ de\ trabalho/teste/programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[gusta@GPArchTP teste]$ █

```

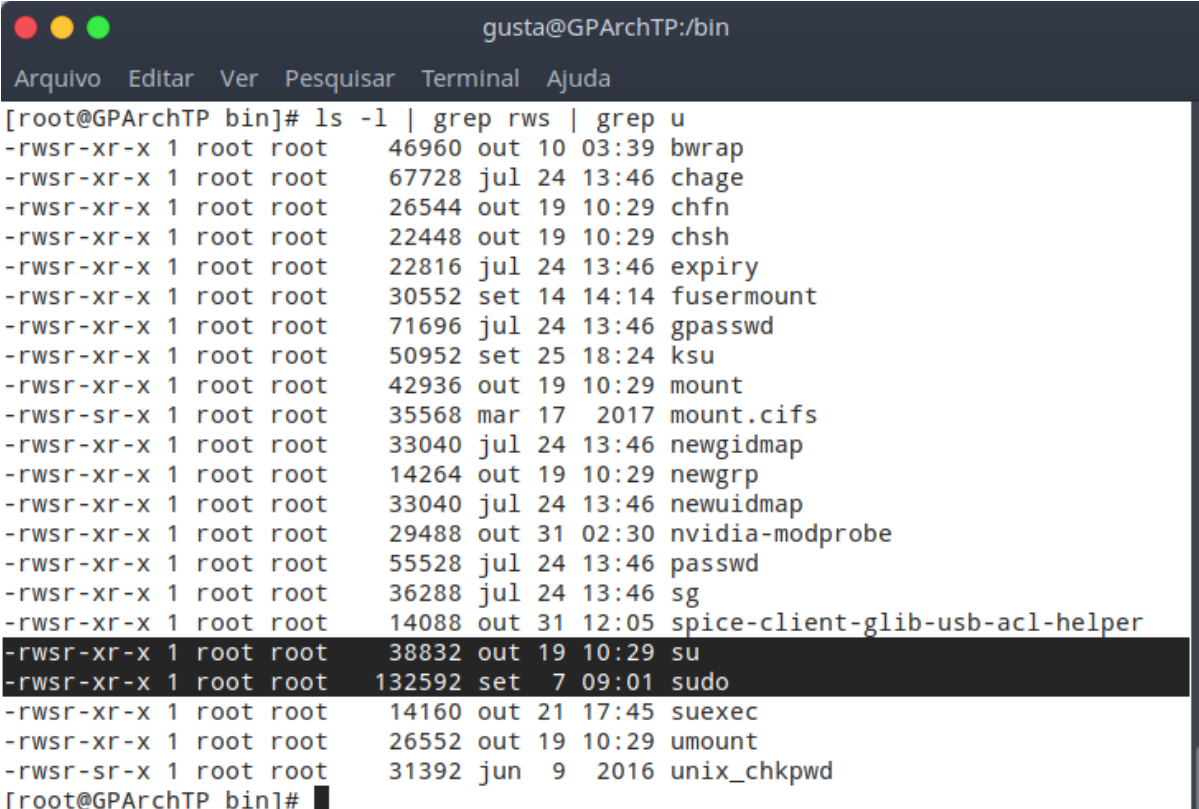
Fonte: O próprio autor.

Outro teste é realizado e o usuário root altera o conteúdo do arquivo. Em seguida, mesmo não conseguindo visualizar o arquivo com vias normais (cat), ao utilizar o programa com o SUID definido, o usuário gusta é capaz de visualizar o conteúdo do arquivo novamente. Porém, quando o usuário root remove o SUID do binário executável do programa (chmod u-s programa), conforme visto com o comando “ls -l”, o usuário gusta deixa de ser capaz de visualizar o conteúdo do arquivo mesmo utilizando o

programa, já que a execução do programa volta a ser realizada com os privilégios de `gusta` ao invés de privilégios de `root`.

Como é possível ver na Figura 7 e nos exemplos utilizando o programa da Figura 5, é possível definir o SUID para qualquer outro programa além do “`sudo`” e do “`su`”.

Figura 7 - Alguns programas no diretório “`/bin`” com SUID definido.



```

gusta@GPArchTP:/bin
Arquivo Editar Ver Pesquisar Terminal Ajuda
[root@GPArchTP bin]# ls -l | grep rws | grep u
-rwsr-xr-x 1 root root 46960 out 10 03:39 bwrap
-rwsr-xr-x 1 root root 67728 jul 24 13:46 chage
-rwsr-xr-x 1 root root 26544 out 19 10:29 chfn
-rwsr-xr-x 1 root root 22448 out 19 10:29 chsh
-rwsr-xr-x 1 root root 22816 jul 24 13:46 expiry
-rwsr-xr-x 1 root root 30552 set 14 14:14 fusermount
-rwsr-xr-x 1 root root 71696 jul 24 13:46 gpasswd
-rwsr-xr-x 1 root root 50952 set 25 18:24 ksu
-rwsr-xr-x 1 root root 42936 out 19 10:29 mount
-rwsr-sr-x 1 root root 35568 mar 17 2017 mount.cifs
-rwsr-xr-x 1 root root 33040 jul 24 13:46 newgidmap
-rwsr-xr-x 1 root root 14264 out 19 10:29 newgrp
-rwsr-xr-x 1 root root 33040 jul 24 13:46 newuidmap
-rwsr-xr-x 1 root root 29488 out 31 02:30 nvidia-modprobe
-rwsr-xr-x 1 root root 55528 jul 24 13:46 passwd
-rwsr-xr-x 1 root root 36288 jul 24 13:46 sg
-rwsr-xr-x 1 root root 14088 out 31 12:05 spice-client-glib-usb-acl-helper
-rwsr-xr-x 1 root root 38832 out 19 10:29 su
-rwsr-xr-x 1 root root 132592 set 7 09:01 sudo
-rwsr-xr-x 1 root root 14160 out 21 17:45 suexec
-rwsr-xr-x 1 root root 26552 out 19 10:29 umount
-rwsr-sr-x 1 root root 31392 jun 9 2016 unix_chkpwd
[root@GPArchTP bin]#

```

Fonte: o próprio autor.

A fim de obter-se esclarecimentos sobre permissões SUID, o apêndice C mostra o processo de atribuição e visualização do SUID.

Araújo (2017) afirma: “Como especialista em tecnologia, eu afirmo: não existe nada 100% seguro quando o assunto é meio digital”, assim como Nemeth (2012, p. 463) também afirma: “Não, o Linux não é seguro. Assim como nenhum outro sistema operacional que se comunica via rede.”.

Seguindo o raciocínio dos autores Araújo (2017) e Nemeth et al. (2012, p. 463), não existe *software* livre de vulnerabilidades. Assim, como o usuário `root` tem privilégios para executar qualquer ação em um sistema operacional Linux, não é saudável utilizar o

SUID para conferir privilégios de root aos programas (exceto programas intrínsecos do sistema operacional, como o “sudo” ,o “su” e outros cuja responsabilidade sobre as ações passa a ser do usuário).

Caso um *hacker* obtenha o controle ilícito de um programa vulnerável que está sendo executado com privilégios de root, o *hacker* poderá explorar toda a máquina. Sendo função desta máquina tratar ou disponibilizar informações e dados sensíveis de uma organização, qualquer ação ilegítima cometida pelo *hacker* pode comprometer toda a organização.

Para evitar esta situação e conferir apenas privilégios necessários para o funcionamento legítimo de um programa, faz-se o uso de *capabilities*.

3.4 **Capabilities**

Usualmente administradores de sistemas instalam *softwares* e definem um usuário particular que executará cada um destes *softwares*, a fim de não conferir a eles privilégios de root sobre o sistema. No entanto, alguns processos necessitam acessar certos recursos do sistema operacional os quais necessitam de privilégios elevados.

Quando estes privilégios referem-se a permissões de manipulação de arquivos e não envolvem o usuário root e as permissões comuns do DAC não atendem a necessidade, torna-se viável a utilização do SUID para alterar o EUID do processo, fazendo com que o DAC forneça o acesso aos arquivos.

No entanto, quando referem-se a outros tipos de recursos do sistema, os privilégios necessitam ser tratados pelo Linux Kernel, que os separa em duas categorias: processos privilegiados e processos não privilegiados.

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes (whose effective user ID is 0, referred to as superuser or root), and unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list). Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. Capabilities are a per-thread attribute (THE LINUX MAN-PAGES PROJECT, 2017).

Os processos privilegiados são todos aqueles que executam com EUID igual a zero, ou seja, ou foram executados pelo usuário root, ou o SUID foi utilizado em sua execução. Estes processos logo ao serem executados terão permissão para manipular qualquer recurso do sistema. Já os processos não privilegiados estão sujeitos a checagem de permissões.

Como as permissões tradicionais do DAC não atendem a todas as necessidades de controle de acesso, a fim de evitar concessão de privilégios desnecessários, foram criadas extensões para o modelo. Entre estas extensões (*acl's*, *namespaces*, entre outras), será abordada uma extensão que permite controle de privilégios com base nas capacidades que um processo possui no sistema (THE LINUX FOUNDATION, 2013).

Estas permissões são segregadas de acordo com os recursos que os processos poderão manipular quando às possuírem. O conjunto destas permissões é conhecido como *capabilities*.

Utilizando como exemplo o conjunto de *capabilities* “CAP_DAC_OVERRIDE”, um processo que o possua é capaz de ler e executar qualquer arquivo do sistema de arquivos sem que o usuário referenciado em seu EUID possua estas permissões concedidas pelo modelo tradicional de DAC (THE LINUX MAN-PAGES PROJECT, 2017) – é graças a atribuição deste conjunto de *capabilities* aos processos privilegiados, que o DAC em seu modelo tradicional é incapaz de restringir qualquer ação do usuário root.

Porém, um processo que possua apenas “CAP_DAC_OVERRIDE” não é capaz de gerenciar as portas de rede com número menor que “1024”. Logo um processo referente a um *WebServer* que trabalhe na porta “80”, por exemplo, necessitaria possuir o conjunto de *capabilities* “CAP_NET_BIND_SERVICE” - *capabilities* responsáveis por gerenciar o acesso a estas portas (THE LINUX MAN-PAGES PROJECT, 2017).

Apesar de extensões do DAC - como *capabilities* - permitirem segregar a concessão de privilégios com maiores restrições, brechas nos programas ainda podem expor todas as funções que estas restrições não abrangem (por exemplo, um *WebServer* com *capabilities* “CAP_NET_BIND_SERVICE” é capaz de abrir todas as portas baixas, não sendo possível restringir esta capacidade apenas à porta 80).

Assim, para tornar o sistema operacional menos vulnerável trazendo maiores possibilidades à filtragem de privilégios, utiliza-se o conceito de controle de acesso mandatório, qual será abordado na seção 3.5.

3.5 Controle Acesso Mandatório

Controle de acesso mandatório, ou MAC, é um conceito que garante o endurecimento da segurança em sistemas operacionais. Devido a segurança proporcionada por este modelo de controle de acesso, ele é frequentemente utilizado por instituições governamentais e instalações militares. Com ele, é possível restringir acesso a um recurso do sistema ainda que o DAC permita a um processo acessá-lo.

Often employed in government and military facilities, mandatory access control works by assigning a classification label to each file system object. Classifications include confidential, secret and top secret. Each user and device on the system is assigned a similar classification and clearance level. When a person or device tries to access a specific resource, the OS or security kernel will check the entity's credentials to determine whether access will be granted. While it is the most secure access control setting available, MAC requires careful planning and The LSM API allows different security models to be plugged into the kernel—typically access control frameworks. To ensure compatibility with existing applications, the LSM hooks are placed so that the Unix DAC checks are performed first, and only if they succeed, is LSM code invoked.continuous monitoring to keep all resource objects' and users' classifications up to date (ROUSE, 2013).

Ao invés de checar permissões da forma tradicional, é realizada uma filtragem avançada de privilégios sem levar em conta apenas o DAC e suas extensões. Como existe uma diversidade de módulos de controle de acesso mandatório, então essas possibilidades de filtragem de privilégios variam de acordo com o módulo utilizado.

Embora exista essa diversidade de módulos, todos eles utilizam uma técnica chamada *labelling*, que consiste na aplicação de *Security Labels*. Esta técnica visa aplicar rótulos em cada objeto do sistema. Assim, é possível atribuir varias informações adicionais aos recursos, possibilitando uma filtragem de privilégios mais fina (THE LINUX FOUNDATION, 2013).

Com esta filtragem mais fina, apenas ações expressamente permitidas pelo administrador do sistema podem ser realizadas pelos programas. Do contrário estas ações serão barradas, ainda que sejam legítimas e o programa esteja sendo executado com EUID de usuário root.

Portanto, devido à forma com que este modelo de controle de acesso funciona, sua aplicação exige planejamento detalhado e pontual, a fim de evitar sérios problemas, como por exemplo, inutilização de funcionalidades do sistema.

Assim, fica claro que um sistema baseado em MAC tem como característica principal a possibilidade de oferecer ao utilizador elevados níveis de segurança. Como a implementação desse modelo de controle de acesso pode ser feita através de módulos distintos, faz-se necessária a utilização de uma API que permita carregar qualquer módulo de segurança, de acordo com a escolha do administrador. Os sistemas Linux, a partir da versão 2.6 do Linux Kernel, passaram a ter nativamente esta API, que é chamada LSM. O conceito de LSM é abordado no capítulo 4.

4 LINUX SECURITY MODULES

LSM (Linux Security Modules) é uma API (*Application Programming Interface*) que permite a implementação de diversos modelos de controle de acesso em um sistema Linux. Esta API fornece abstração em nível do Linux Kernel.

Em 2001, na Linux Kernel Summit, a NSA (Agência de Segurança Nacional Americana) apresentou seus trabalhos sobre um mecanismo de controle de acesso mandatório denominado SELinux e sugeriu que ele fosse incluído na versão 2.5 do Linux Kernel como meio de controle de acesso mandatório padrão. Torvalds, o mentor do Linux reprovou a ideia fazendo uma série de considerações que um *framework* de segurança deveria ter para ser incluído no Linux Kernel, descrevendo este *framework* como um meio de permitir implementar ao sistema qualquer modelo de controle de acesso desejado (SMALLEY, FRASER E VANCE, [s.d.]).

Em resposta a isso, Crispin Cowan sugeriu a criação de uma plataforma que fornecesse interceptações de chamadas de sistema de forma satisfatória a tornar capaz a implementação de mecanismos de controle de acesso mandatório como módulos adicionais que podem ser adquiridos e carregados de acordo com as necessidades de cada utilizador dos sistemas Linux, atendendo assim as considerações feitas por Torvalds e tornando a LSM aceitável para ser implementada no Linux Kernel.

One of the byproducts of the Linux 2.5 Kernel Summit <http://lwn.net/2001/features/KernelSummit/> was the notion of an enhancement of the loadable kernel module interface to facilitate security-oriented kernel modules. The purpose is to ease the tension between folks (such as Immunix and SELinux) who want to add substantial security capabilities to the kernel, and other folks who want to minimize kernel bloat & have no use for such security extensions. Modules that can be loaded, or not, are the obvious solution, but the current LKM does not export sufficient hooks to support many security mechanisms. Thus many current security enhancements end up existing as kernel patches, which marginalizes their utility by making distribution problematic. The proposed solution is to enhance the LKM with a variety of new kernel elements exported to the module interface, so as to support a reasonable variety of security enhancements. We have started a new mailing list called linux-security-module. The charter is to design, implement, and maintain suitable enhancements to the LKM to support a reasonable set of security enhancement packages. The prototypical module to be produced would be to port the POSIX Privs code out of the kernel and make it a module. An essential part of this project will be that the resulting work is acceptable for the mainline Linux kernel (COWAN, 2001).

Foi criado assim o projeto LSM, que levou mais de dois anos para ser desenvolvido. Além dos esforços da comunidade LSM, houveram grandes contribuições

de colaboradores de outros projetos como: Immunix, SGI, SELinux e Janus (SMALLEY, FRASER E VANCE, [s.d.]). Em dezembro de 2003, finalmente o projeto estava concluído e a API LSM foi então incluída na versão oficial do Linux Kernel 2.6.

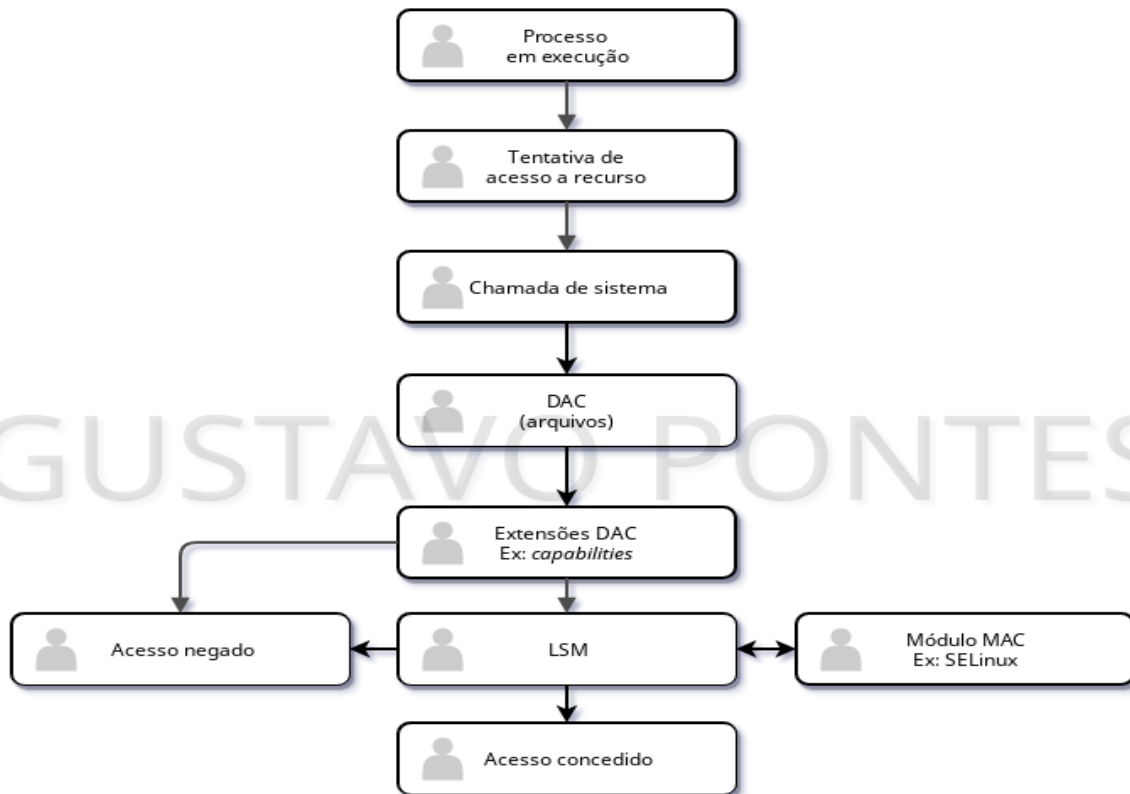
4.1 Abstração fornecida pela LSM

A abstração fornecida pela LSM, permite a utilização de módulos adicionais de segurança em um sistema Linux. Como dito anteriormente, esta API trabalha fornecendo abstração entre o módulo de segurança instalado e o Linux Kernel. Isso é possível através de interceptação de chamadas de sistema.

The LSM API allows different security models to be plugged into the kernel—typically access control frameworks. To ensure compatibility with existing applications, the LSM hooks are placed so that the Unix DAC checks are performed first, and only if they succeed, is LSM code invoked (THE LINUX FOUNDATION, 2013).

Sempre que um programa realiza uma tentativa de acesso a um recurso do sistema, é gerada uma chamada de sistema. Esta chamada de sistema segue o fluxo apresentado na Figura 8.

Como é possível observar, os módulos MAC instalados são consultados pela LSM, que está em mais baixo nível que o DAC e suas extensões. Portanto os módulos LSM são a última instância a ser consultada antes do acesso ser concedido pelo Linux Kernel. No entanto, se o DAC e sua extensões não concederem o acesso, a chamada de sistema não chega a ser repassada para a LSM.

Figura 8 – Fluxo das chamadas de sistema

Fonte: o próprio autor.

4.2 Exemplos de módulos LSM

Devido à liberdade intrínseca dos sistemas de código aberto, há uma grande variedade de módulos de controle de acesso que podem ser implementados com a LSM. Estes devem ser escolhidos com base nas necessidades da política de segurança empregada ao sistema. Embora haja esta liberdade, alguns módulos são os aceitos oficialmente. Alguns dos módulos mais conhecidos são:

- AppArmor: Primariamente desenvolvido com o nome “SubDomain” para integrar a distribuição Linux WireX (Immunix), porém o projeto abandonou a distribuição e foi portado para a distribuição SuSe Linux. Posteriormente foi adquirido pela Novell em 2005 e renomeado para AppArmor. Em 2009 a Canonical obteve os direitos de desenvolvimento e manutenção, os quais detém até o momento (APPARMOR, 2012).
- SELinux: Desenvolvido originalmente pela NSA com apoio de colaboradores de outras grandes companhias como a Red Hat Enterprises (NSA, 2016).

- Tomoyo: Lançado em março de 2003, não só é um módulo totalmente funcional de controle de acesso mandatório, como também uma ferramenta de análise do sistema, além disso sua utilização é fácil. Desde março de 2012 o projeto é patrocinado pela NTT Data Corporation (TOMOYO, 2017).

Devido ao fato do SELinux ter sido criado pela NSA e distribuído nos sistemas Red Hat Enterprises, o mesmo é o módulo abordado no capítulo 5. A distribuição Linux utilizada no cenário de testes é a CentOS, dado o fato de ser baseada no sistema da Red Hat Enterprises.

5 SELINUX

Security-Enhanced Linux é o nome dado a um módulo de segurança do Linux Kernel, cujo qual fornece um mecanismo completo de controle de acesso mandatório. O projeto foi originalmente desenvolvido pela NSA, embora tenha recebido contribuições de empresas do mercado de segurança, como a Red Hat Company, que mantém o SELinux até o presente como solução comercial.

Devido sua ligação com a Agência Nacional de Segurança Americana, este foi desenvolvido com base no modelo de controle de acesso utilizado pelo Departamento de Defesa dos Estados Unidos.

O SELinux é um produto do governo e da indústria, com design e desenvolvimento fornecidos pela NSA, Network Associates, Tresys e muitas outras. Embora tenha sido introduzido pela NSA como um conjunto de correções, foi inoculado no *kernel* Linux 2.6 (JONES, 2008).

Sua primeira versão, foi lançada em primeiro de janeiro de 1998 sobre o licenciamento GNU GPL, que permite livre distribuição e desenvolvimento comunitário.

Embora a implementação atual do SELinux seja baseada na LSM, a criação deste projeto precede o lançamento da API, visto que a LSM somente foi idealizada e lançada anos depois, em dezembro de 2003 como resposta à rejeição de Linus Torvalds na Linux Kernel Summit de 2001 sobre a inclusão do SELinux na versão 2.5 do Linux Kernel como principal meio de controle de acesso mandatório (SMALLEY, FRASER E VANCE, [s.d.]).

O objetivo deste módulo é a criação de um ambiente computacional baseado em Linux, cujo qual é capaz de manter pilares de segurança como confidencialidade e integridade das informações nele contidas de forma rígida e que possa abranger as mais variadas customizações.

Para garantir a segurança, o SELinux trata usuários, arquivos, dispositivos, processos e serviços aplicando a eles *Security Labels*. Estes rótulos de segurança permitem limitar a interação entre objetos do sistema através de regras definidas pelo administrador.

5.1 Maneiras de restrição com SELinux

O SELinux possui os seguintes modos de operação (THE CENTOS PROJECT, 2017):

- *Enforcing*: Restringe todas as ações não previstas nos módulos de permissão e registra em log cada uma destas ações.
- *Permissive*: a função do modo permissivo é permitir todas as ações dos processos, porém ainda registra em log cada ação que viole uma política de isolamento.
- *Disabled*: O SELinux não realiza nenhum bloqueio, nem registra eventos em log.

No modo *enforcing*, o SELinux é capaz de trabalhar com diferentes políticas, sendo as mais utilizadas: “*Strict*”, que por padrão confina e isola cada objeto do sistema. Assim, para que qualquer objeto possa interagir com qualquer outro, é necessário que exista uma regra autorizando esta interação; “*Targeted*”, que isola apenas um grupo de processos predefinidos (THE CENTOS PROJECT, 2017).

Trabalhando com a política *Targeted*, o SELinux é capaz de trabalhar com os seguintes modelos de restrição:

- *Type Enforcement*: trata os recursos do sistema como objetos e de forma individual. Para que um processo possa interagir com outro recurso do sistema, é necessário que ambos possuam rótulos definindo o contexto destes e uma política que permita a interação entre estes contextos.
- *Multi Category Security* ou MCS: neste modelo as permissões são tratadas por categorias, assim permitir interações em larga escala torna-se mais prático pela possibilidade de agrupar objetos com rótulos semelhantes.
- MLS ou *Multi Level Security*: é um modelo que baseia-se em instalações militares e órgãos governamentais, como a própria NSA, precursora do SELinux. Este modelo permite definir níveis de acesso.

No modelo *Type Enforcement*, tomando como exemplo o almoxarifado de um hospital, recepcionistas são rotulados como profissionais de escritório e médicos e

enfermeiros são rotulados como profissionais de saúde. Dentro do almoxarifado do hospital há duas salas, uma para materiais de escritório e outra de materiais de saúde. A diretoria do hospital estabelece uma regra definindo que apenas os profissionais de saúde poderão ter acesso à sala de materiais de saúde e que apenas os profissionais de escritório poderão ter acesso à sala de materiais de escritório. Assim, os médicos e enfermeiros poderão retirar objetos como luvas esterilizadas e os outros objetos contidos na sala de materiais de saúde, mas não folhas sulfite e outros objetos da sala de materiais de escritório; e os recepcionistas poderão retirar folhas sulfite e outros materiais de escritório, mas não luvas esterilizadas e outros materiais de saúde.

Ainda utilizando o exemplo do almoxarifado hospitalar, é possível explicar o modelo MCS. Através da segregação por categorias, é possível definir que dentro da sala de materiais de saúde onde há uma prateleira rotulada como materiais de enfermagem e outra rotulada como materiais de medicina, ainda que ambos sejam profissionais de saúde e tenham acesso à sala de materiais de saúde, apenas enfermeiros possam retirar materiais de enfermagem e apenas médicos podem retirar os materiais de medicina, enquanto recepcionistas sequer podem ter acesso à sala.

Para explicar o modelo MLS, toma-se como exemplo a aplicação em uma agência governamental de segurança. Este modelo permite definir que um agente de patente nível 3 tenha acesso a todos os documentos *top secret* (nível 3), *secretos* (nível 2) e não classificados (nível 1), enquanto um agente de patente nível 2 tem acesso a todos os documentos *secretos* e caracterizados, mas não aos *top secret*, ao passo que um agente de nível 1 tem acesso apenas aos documentos não classificados.

Quando operando com a política *Strict*, o SELinux é capaz de trabalhar não apenas com os modelos citados anteriormente, mas também com o modelo RBAC (*Role Based Access Control*), que é útil para estabelecer regras com base nas ações e funções que os usuários representarão dentro do sistema.

Estes modelos, apesar de tratar as interações entre objetos de maneiras distintas, podem ser utilizados em conjunto, tornando maior o nível de proteção. Contudo, na política *Targeted*, o *Type Enforcement* é o modelo mais utilizado (THE CENTOS PROJECT, 2017), desta forma, este será o modelo abordado nas seções seguintes.

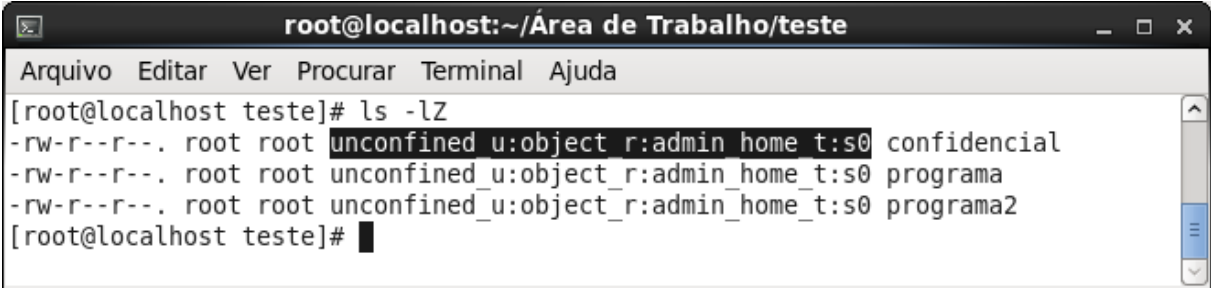
5.2 Conceitos técnicos

Para que seja possível estabelecer um sistema com segurança funcional utilizando SELinux, é indispensável entender alguns conceitos técnicos, como a estrutura dos rótulos, a forma com que as políticas tratam os rótulos, a criação e manutenção das regras e o diagnóstico em situações onde uma funcionalidade de um programa é bloqueada indevidamente.

5.2.1 Rótulos

Os rótulos definem contextos onde os objetos se encontram. Estes rótulos são aplicados sobre todos os recursos do sistema quando o SELinux está ativo e desde então é possível visualizá-los a partir do comando “ls” junto ao argumento “-Z”, vide Figura 9:

Figura 9 - comando “ls -lZ”.



```

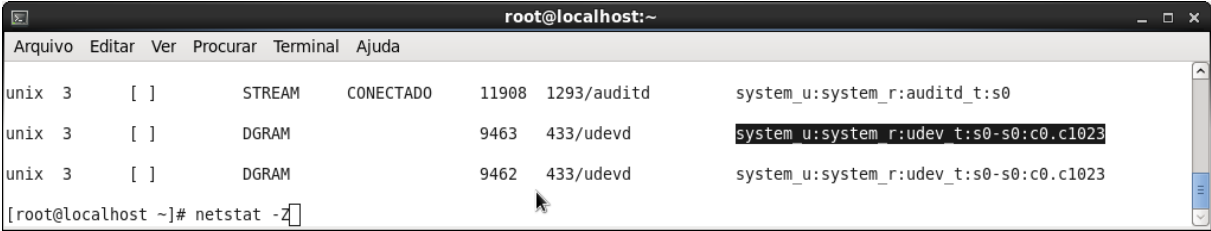
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -lZ
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]#

```

Fonte: o próprio autor.

É importante ressaltar que não apenas os arquivos possuem rótulos, mas também todos os outros recursos do sistema, como processos, *sockets*, portas, usuários, dispositivos e qualquer outro objeto manipulado pelo sistema operacional, como é possível ver nas Figuras 10, 11 e 12:

Figura 10 - comando “netstat -Z”.



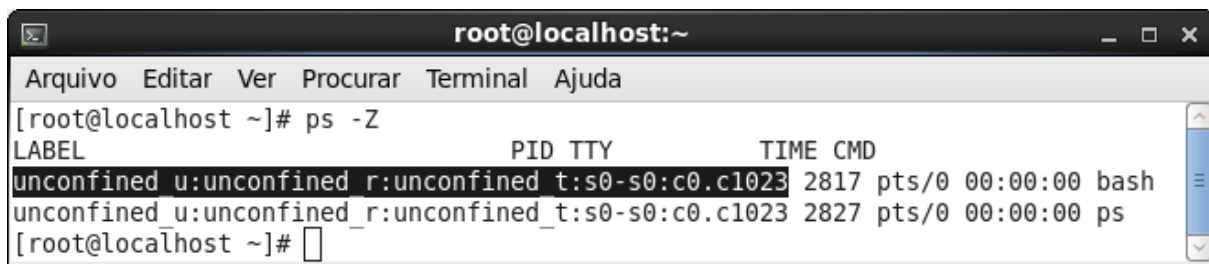
```

root@localhost:~
Arquivo Editar Ver Procurar Terminal Ajuda
unix 3 [ ] STREAM CONECTADO 11908 1293/auditd system_u:system_r:auditd_t:s0
unix 3 [ ] DGRAM 9463 433/udev system_u:system_r:udev_t:s0-s0:c0.c1023
unix 3 [ ] DGRAM 9462 433/udev system_u:system_r:udev_t:s0-s0:c0.c1023
[root@localhost ~]# netstat -Z

```

Fonte: o próprio autor.

Figura 11 - comando “ps -Z”.



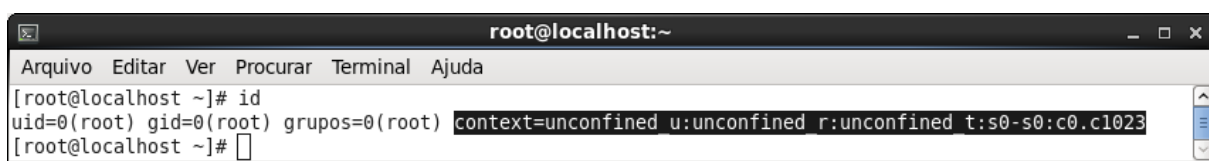
```

root@localhost:~
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost ~]# ps -Z
LABEL                                PID TTY          TIME CMD
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2817 pts/0 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2827 pts/0 00:00:00 ps
[root@localhost ~]#

```

Fonte: o próprio autor.

Figura 12 - comando “id”.



```

root@localhost:~
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost ~]# id
uid=0(root) gid=0(root) grupos=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@localhost ~]#

```

Fonte: o próprio autor.

Na Figura 9, após as permissões comuns do sistema de arquivos, encontra-se o rótulo SELinux. Este rótulo define o contexto onde o arquivo se encaixa e segue a estrutura “*User:Role:Type:Level*”, que é padrão para todos os rótulos aplicados independentemente de qual seja o objeto do sistema, conforme vê-se nas Figuras 10, 11 e 12.

O atributo “*User*” trata do usuário SELinux que terá permissão para manipular o objeto.

“*Role*” é uma implementação de domínio do usuário dentro do sistema. Com esse atributo é possível definir quais são as funções que o usuário pode ter acesso, como por exemplo, funções administrativas ou comuns.

Os atributos “*User*” e “*Role*” são utilizados pela política Strict, visto que são utilizados pelo modelo RBAC.

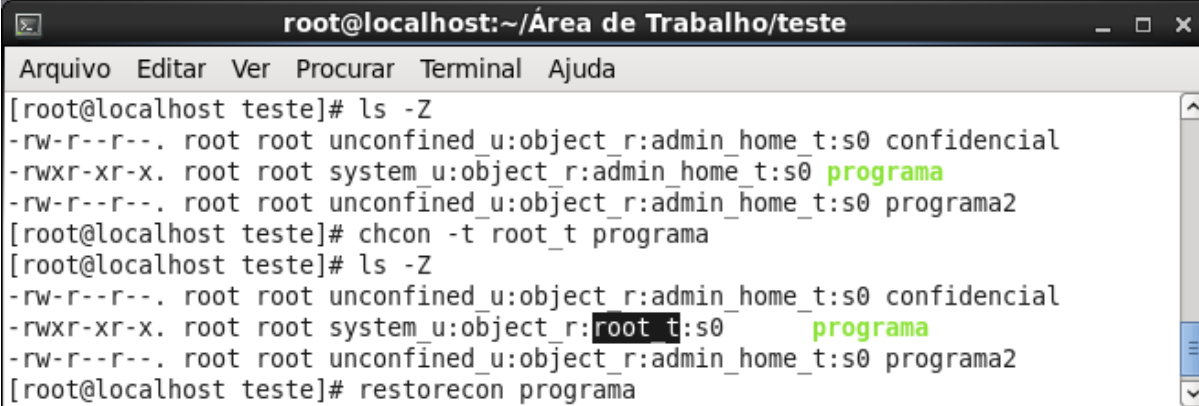
“*Type*” trata-se de um atributo que permite uma espécie de segregação por categorias. Desta forma, passa a ser extremamente útil para o modelo *Type Enforcement*, sendo possível relacionar facilmente processos com recursos que podem ser acessados por eles.

Por último, o atributo “*Level*” permite definir o nível de acesso qual é necessário para manipular o objeto, conforme abordado na explanação do modelo MLS (seção 5.1).

Exclusivamente por questões de conveniência, os valores dos atributos sempre terminam com a primeira letra do nome do atributo em questão, grafada em sua forma minúscula. Assim, os valores de *User*, *Role*, *Type* e *Level* sempre terminarão com “_u”, “_r”, “_t” e “_l”, respectivamente.

Para alterar um rótulo temporariamente – normalmente para fins de testes - utiliza-se o comando “chcon”, que altera temporariamente o contexto do objeto, vide Figura 13:

Figura 13 - comando “chcon -t root_t programa”.



```

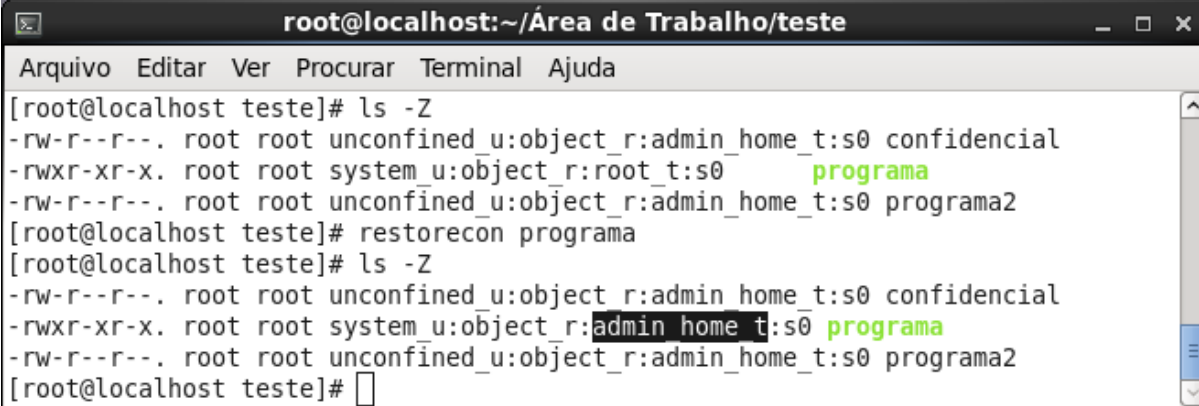
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:admin_home_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# chcon -t root_t programa
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:root_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# restorecon programa

```

Fonte: O próprio autor.

Quando faz-se necessário restaurar o rótulo ao contexto padrão, então utiliza-se o comando “restorecon”, que buscará nas definições do SELinux o contexto padrão para aquele objeto e o aplicará, conforme vê-se na Figura 14.

Figura 14 - comando “restorecon programa”.



```

root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:root_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# restorecon programa
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:admin_home_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]#

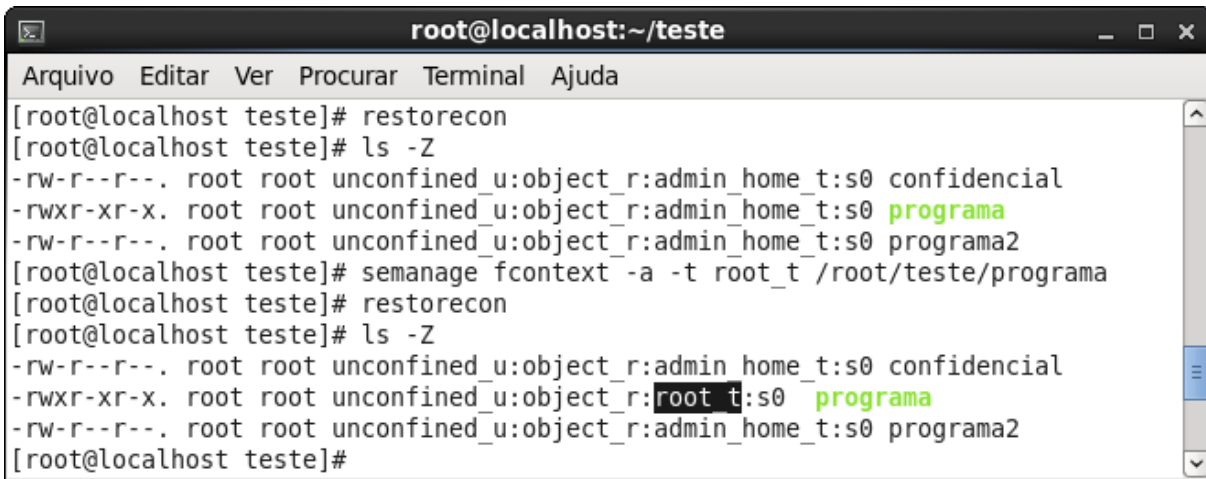
```

Fonte: O próprio autor.

Para alterar permanentemente o rótulo de um objeto, conforme vê-se na Figura 15, faz-se o uso do comando “semanage fcontext”, que é capaz de alterar o contexto padrão

para o objeto nas definições do SELinux, desta forma, sempre que forem consultadas, o valor definido será aplicado ao objeto.

Figura 15 - comando “semanage fcontext -a -t root_t programa”.



```

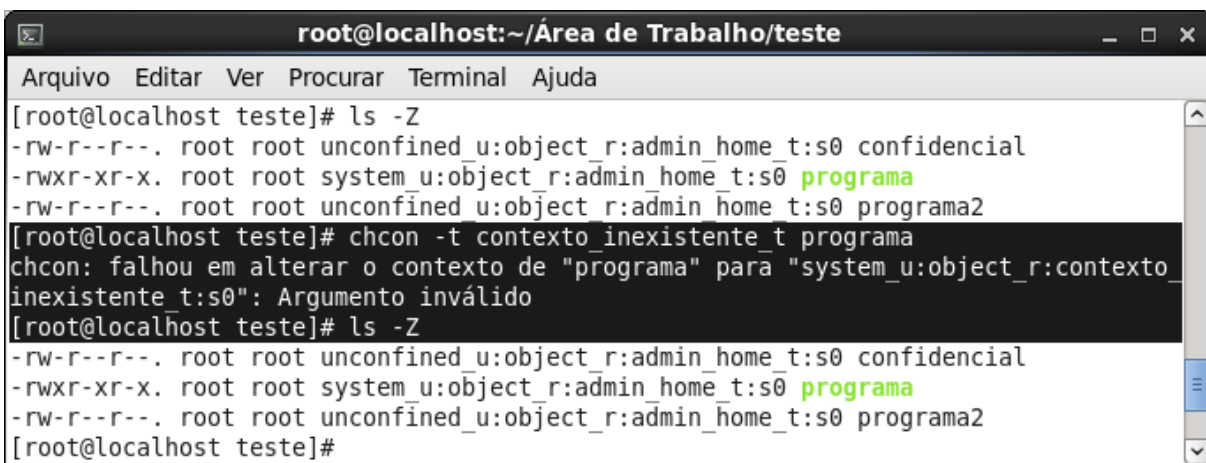
root@localhost:~/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# restorecon
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root unconfined_u:object_r:admin_home_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# semanage fcontext -a -t root_t /root/teste/programa
[root@localhost teste]# restorecon
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root unconfined_u:object_r:root_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]#

```

Fonte: O próprio autor.

Ponto importante a se esclarecer é que um valor pode ser atribuído ao rótulo apenas se o contexto referente a este valor estiver declarado em algum módulo do SELinux, caso contrário uma mensagem de erro será exibida, conforme visto na Figura 16.

Figura 16 - erro ao tentar definir um contexto inválido.



```

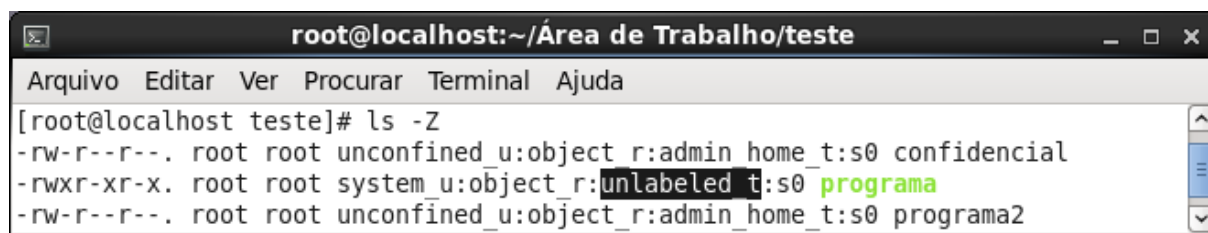
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:admin_home_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# chcon -t contexto_inexistente_t programa
chcon: falhou em alterar o contexto de "programa" para "system_u:object_r:contexto_inexistente_t:s0": Argumento inválido
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:admin_home_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]#

```

Fonte: o próprio autor.

Também vale ressaltar, que caso o módulo onde encontra-se declarado o contexto de um dos atributos – *User*, *Role*, *Type* ou *Level* - for removido do SELinux ou desativado, todos os rótulos de objetos que continham este contexto serão alterados para “unlabeled”, como é possível ver na Figura 17:

Figura 17 - remoção de contexto por deleção ou desativação de módulo.



```
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:unlabeled_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
```

Fonte: O próprio autor.

5.2.2 Type Enforcement - Regras

As regras são as diretrizes que o SELinux irá utilizar para restringir ou permitir as interações entre os objetos do sistema, através delas cria-se uma relação de permissão de acesso entre os contextos destes objetos. No modelo *Type Enforcement*, as regras são baseadas na definição de contextos onde o único atributo considerado é o *Type*.

Na Figura 9 é possível ver que o atributo *Type* sempre está assumindo o valor “admin_home_t”. Isto ocorre porque estes objetos foram criados dentro de um diretório com o mesmo contexto – portanto herdaram o contexto do diretório pai - e significa que os objetos em questão estão sendo confinados e podem ser acessados apenas pelos processos cujo contexto possui permissão para acessar e manipular os arquivos do contexto “admin_home_t”.

A criação destas regras é feita a partir do desenvolvimento de módulos SELinux, cujos quais declaram os contextos que os atributos dos rótulos podem assumir e definem a relação de permissões entre objetos que possuem estes contextos e objetos de outros contextos. Ex: As permissões de um processo cujo *Type* é igual a “x_t” sobre um arquivo com *Type* igual a “y_t”.

O apêndice D demonstra a estrutura de um módulo SELinux.

6 CENÁRIOS

A fim de exemplificar o funcionamento do SELinux, a seção 6.1 tratará do primeiro cenário de testes, onde o SELinux é responsável por controlar a execução do programa da Figura 5.

A seção 6.2 retrata um cenário onde é necessária adição de permissões para um processo, posteriormente à inclusão de seu módulo no SELinux.

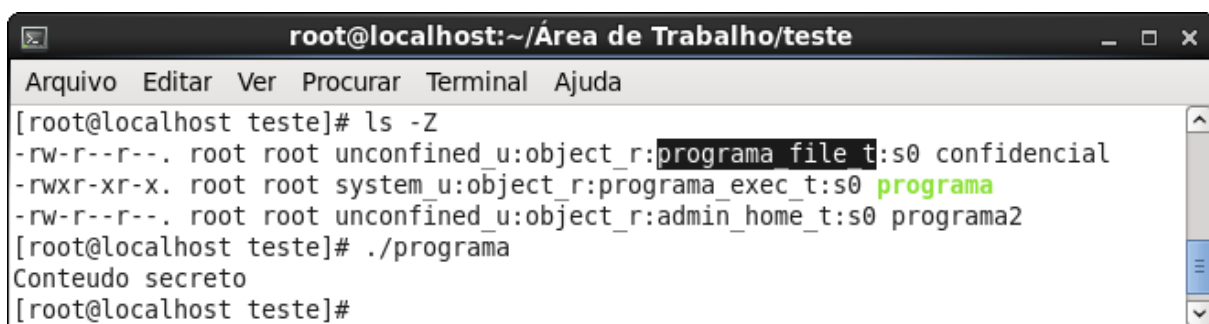
A seção 6.3 retrata um cenário fictício, porém facilmente alcançado em situações reais. Este cenário exhibe o exemplo de uma situação onde o SELinux evita quebra da confidencialidade de um dos arquivos críticos do sistema operacional, o arquivo que armazena as senhas da máquina.

A seção 6.4 refere-se à discussão sobre os resultados obtidos nas outras seções deste capítulo.

6.1 Cenário 1: Controlando a execução de um programa

O módulo demonstrado no apêndice D, permite que a execução do programa da Figura 5 seja controlada pelo SELinux, tendo permissão para acessar apenas os recursos do SO necessários para o seu funcionamento e os arquivos com contexto “programa_file_t”, que dizem respeito ao seu funcionamento normal. Assim, quando utilizado para este fim, conforme é possível ver na Figura 18, não há problemas com permissão e o programa realiza sua função corretamente.

Figura 18 - funcionamento normal.

A terminal window titled "root@localhost:~/Área de Trabalho/teste" with a menu bar containing "Arquivo", "Editar", "Ver", "Procurar", "Terminal", and "Ajuda". The terminal output shows the command "ls -Z" and its results: "-rw-r--r--. root root unconfined_u:object_r:programa_file_t:s0 confidencial", "-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa", and "-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2". The command "./programa" is then executed, resulting in the output "Conteúdo secreto".

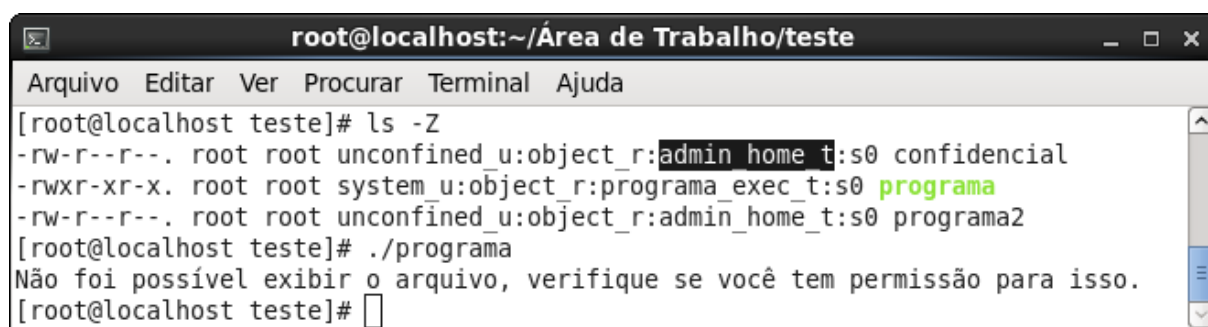
```
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:programa_file_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# ./programa
Conteúdo secreto
[root@localhost teste]#
```

Fonte: o próprio autor.

No entanto, conforme fica visível na Figura 19, quando o programa tenta acessar o conteúdo de um arquivo com outro contexto (podendo até mesmo ser o contexto

“admin_home_t”, pois para este contexto – conforme vê-se no módulo do apêndice D – o processo possui permissão apenas para buscar arquivos nos diretórios, mas não manipulá-los), o programa exibe a mensagem “Não foi possível exibir o arquivo, verifique se você tem permissão para isso”, da mesma forma que ocorreu no exemplo onde não haviam privilégios suficientes de acordo com permissões DAC, em 3.3.1.

Figura 19 - privilégios insuficientes.



```
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# ./programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[root@localhost teste]#
```

Fonte: o próprio autor.

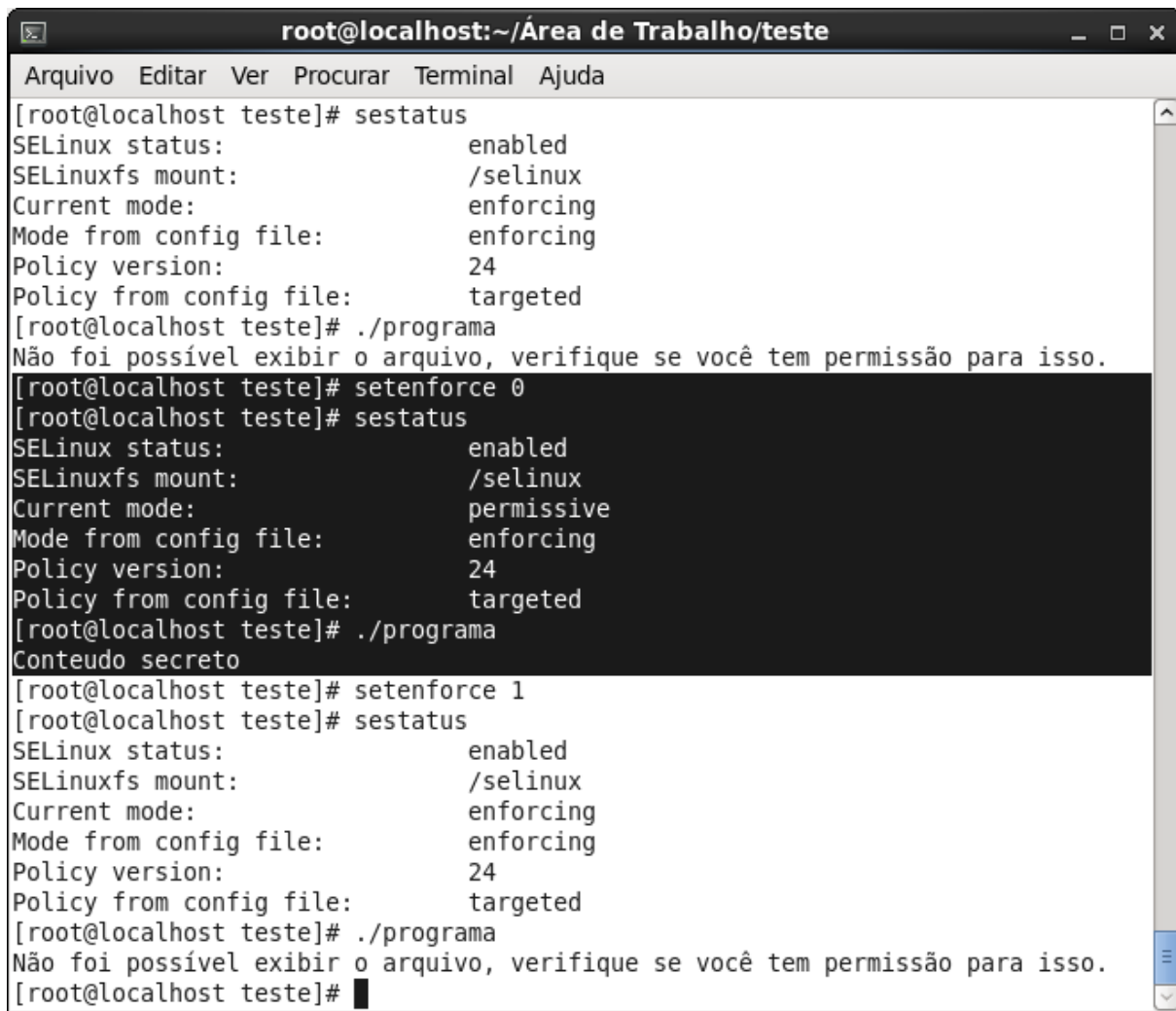
Ainda na Figura 19, é possível ver que o usuário root é quem está executando o programa. Além do root não possuir restrições sobre o sistema e poder manipular qualquer recurso do sistema (princípio básico do DAC, como dito em 3.3), como proprietário do arquivo “confidencial”, o usuário root e qualquer processo executado com seu EUID deveriam conseguir manipular o arquivo em questão - devido às permissões ‘rwxr-xr-x’ atribuídas ao arquivo. No entanto, como dito em 3.5, o modelo MAC é capaz de restringir ações até mesmo do usuário root. Como o módulo do apêndice D não permite aos programas com contexto “programa_t” manipular arquivos com contexto “admin_home_t”, então o acesso é negado.

Os comandos “setenforce 0” e “setenforce 1” fazem que o SELinux entre em modo permissivo e restritivo, respectivamente. O comando “sestatus” é responsável por mostrar informações de funcionamento do SELinux, incluindo o modo em que ele está operando e qual a política que ele está adotando (neste caso a *Targeted*). A Figura 20 exemplifica a utilização destes comandos.

Assim, a fim de comprovar que o SELinux é o responsável por negar o acesso no exemplo da Figura 19, é possível utilizar os comandos mostrados na Figura 20. Para isso, basta verificar que após fazer com que o SELinux entre em modo permissivo, ao executar o programa o conteúdo do arquivo é exibido e após fazer com que o SELinux volte ao

modo restritivo, quando o programa é executado, vê-se novamente a mensagem “Não foi possível exibir o arquivo, verifique se você tem permissão para isso”, indicando assim que o SELinux é realmente eficaz em restringir os privilégios de qualquer usuário, incluindo o root.

Figura 20 - comando “setenforce”.

A terminal window titled "root@localhost:~/Área de Trabalho/teste" showing the execution of the 'setenforce' command. The terminal output is as follows:

```
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# sestatus
SELinux status:                enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
[root@localhost teste]# ./programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[root@localhost teste]# setenforce 0
[root@localhost teste]# sestatus
SELinux status:                enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
[root@localhost teste]# ./programa
Conteudo secreto
[root@localhost teste]# setenforce 1
[root@localhost teste]# sestatus
SELinux status:                enabled
SELinuxfs mount:                /selinux
Current mode:                   enforcing
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
[root@localhost teste]# ./programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[root@localhost teste]#
```

Fonte: o próprio autor.

6.2 Cenário 2: Manipulação de permissões

Apesar da função do programa da figura 5 ser apenas mostrar o conteúdo de um arquivo e o contexto que este arquivo pode assumir estar previsto em seu módulo SELinux, há casos onde será necessário dar permissões a um processo para acessar objetos com outros contextos futuramente a criação de seu módulo principal (módulo retratado no apêndice D).

A fim de demonstrar o processo de concessão de permissões após a criação do módulo principal do programa, será criado e instalado no SELinux um módulo de política adicional para permitir ao programa em questão manipular o arquivos com o contexto “admin_home_t”.

Para viabilizar este processo, devem ser utilizadas algumas ferramentas de gerenciamento do SELinux, como o “semodule”, o “audit2allow” e um *make file* contido em “/usr/share/selinux/devel/Makefile”.

Em modo restritivo, sempre que um programa realiza uma tentativa de acesso a um recurso e o SELinux nega o acesso, é criada uma entrada no arquivo “/var/log/audit/audit.log”. Além disso, o processo não é capaz de executar suas instruções seguintes. Porém, se o SELinux estiver no modo permissivo, o processo executará as próximas instruções criando uma entrada no arquivo de log para cada ação não autorizada.

Portanto, a utilização do modo permissivo do SELinux é bastante útil para solucionar problemas com permissões, já que indica através do arquivo de log, exatamente quais instruções estão sendo bloqueadas de uma só vez, sendo assim possível criar um módulo de regras que permitam o acesso a todos os recursos necessários para o processo de forma prática e rápida.

Vale lembrar que criação de módulos de permissão também é possível no modo restritivo, porém, como a execução é bloqueada toda vez que não há uma regra que permita uma instrução e as instruções seguintes não são executadas (não criando entradas no arquivo de logs) até que um módulo que permita a execução da instrução anterior seja incluído no SELinux, será necessário repetir o processo de criação e instalação até que não existam mais instruções bloqueadas.

Para criar o módulo, é necessário analisar as entradas de log criadas pelo SELinux, que não possuem um formato amigável, conforme é possível ver na Figura 21.

Figura 21 - alertas do SELinux (comando “grep programa /var/log/audit/audit.log | cat”).


```

root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# grep programa /var/log/audit/audit.log | cat
type=AVC msg=audit(1511408953.533:831): avc: denied { read } for pid=8109 comm="progra
ma" name="confidencial" dev=dm-0 ino=684147 scontext=unconfined_u:unconfined_r:programa_t
:s0-s0:c0.c1023 tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file
type=AVC msg=audit(1511408953.533:831): avc: denied { open } for pid=8109 comm="progra
ma" name="confidencial" dev=dm-0 ino=684147 scontext=unconfined_u:unconfined_r:programa_t
:s0-s0:c0.c1023 tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file
type=SYSCALL msg=audit(1511408953.533:831): arch=40000003 syscall=5 success=yes exit=3 a0
=bff4780b a1=0 a2=1b6 a3=8048675 items=0 ppid=7763 pid=8109 auid=0 uid=0 gid=0 euid=0 sui
d=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1 comm="programa" exe=2F726F6F742FC3817265
612064652054726162616C686F2F74657374652F70726F6772616D61 subj=unconfined_u:unconfined_r:p
rograma_t:s0-s0:c0.c1023 key=(null)
type=AVC msg=audit(1511408953.537:832): avc: denied { getattr } for pid=8109 comm="pro
grama" path=2F726F6F742FC3817265612064652054726162616C686F2F74657374652F636F6E666964656E6
369616C dev=dm-0 ino=684147 scontext=unconfined_u:unconfined_r:programa_t:s0-s0:c0.c1023
tcontext=unconfined_u:object_r:admin_home_t:s0 tclass=file
type=SYSCALL msg=audit(1511408953.537:832): arch=40000003 syscall=197 success=yes exit=0
a0=3 a1=bff476f0 a2=94aff4 a3=9f76008 items=0 ppid=7763 pid=8109 auid=0 uid=0 gid=0 euid=
0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=1 comm="programa" exe=2F726F6F742FC38
17265612064652054726162616C686F2F74657374652F70726F6772616D61 subj=unconfined_u:unconfine
d_r:programa_t:s0-s0:c0.c1023 key=(null)
[root@localhost teste]#

```

Fonte: o próprio autor.

No entanto, a ferramenta “audit2allow” pode ser utilizada para facilitar o entendimento, e mais do que isso, fornecer através destas entradas o módulo de regras pronto, em formato semelhante ao módulo retratado no apêndice D, conforme é possível ver na Figura 22:

Figura 22 - Criação de módulo com a ferramenta audit2allow.

```

root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# grep programa /var/log/audit/audit.log | audit2allow -D -M modulo
***** IMPORTANT *****
To make this policy package active, execute:

semodule -i modulo.pp

[root@localhost teste]# cat modulo.te

module modulo 1.0;

require {
    type admin_home_t;
    type programa_t;
    class file { read getattr open };
}

#===== programa t =====
allow programa t admin_home_t:file { read getattr open };
[root@localhost teste]#

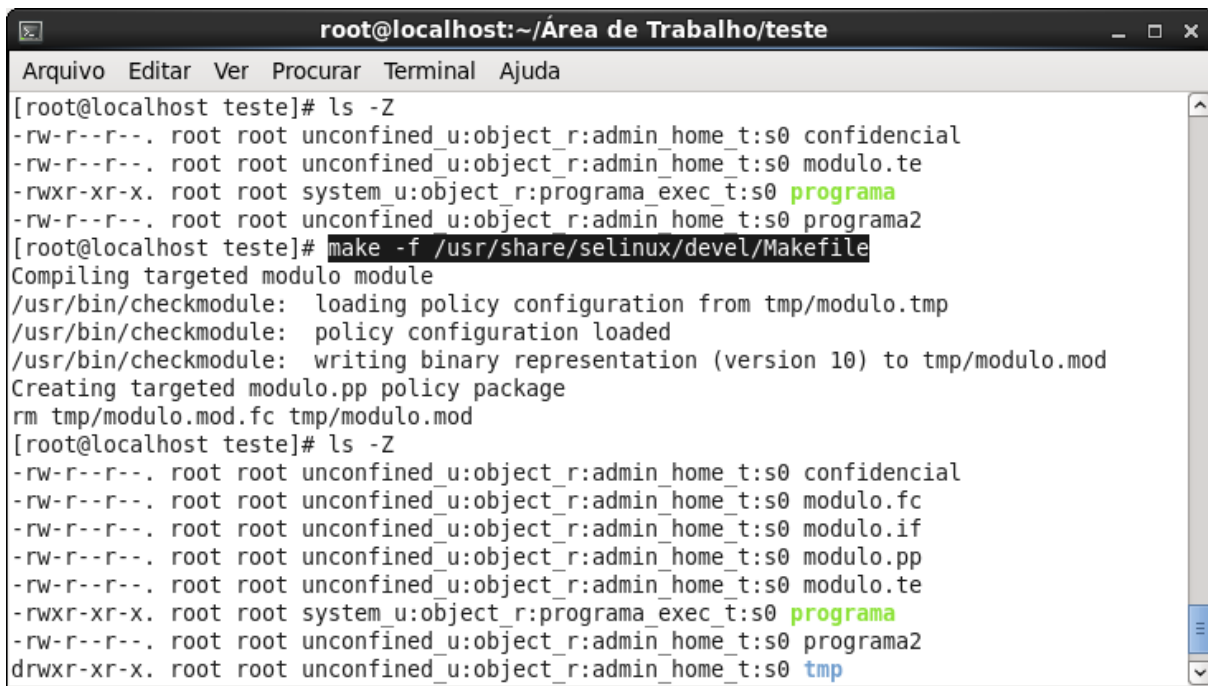
```

Fonte: o próprio autor.

Um módulo SELinux para o modelo *Type Enforcement*, é composto essencialmente de três arquivos, sendo um arquivo com a extensão “.te” (que declara o contexto, as instruções e permissões, conforme explicado no apêndice D), um arquivo com a extensão “.fc” (que define o contexto padrão que os arquivos do programa qual este módulo irá se aplicar deverão assumir) e um arquivo com a extensão “.if” (que fornece uma espécie de API para facilitar o desenvolvimento de outros módulos), embora apenas o arquivo com a extensão “.te” necessite obrigatoriamente possuir conteúdo.

Todos estes devem estar presentes no mesmo diretório, possuir o mesmo nome que foi declarado no arquivo com a extensão “.te”. Assim, com a execução do comando “*make -f /usr/share/selinux/devel/Makefile*” no mesmo diretório, os três arquivos (se apenas o arquivo “.te” existir, o *make* file criará os outros com conteúdo nulo) serão compilados em um arquivo único com o mesmo nome e a extensão “.pp”, que é o módulo SELinux propriamente dito, conforme é possível ver na Figura 23:

Figura 23 - compilação de módulo.

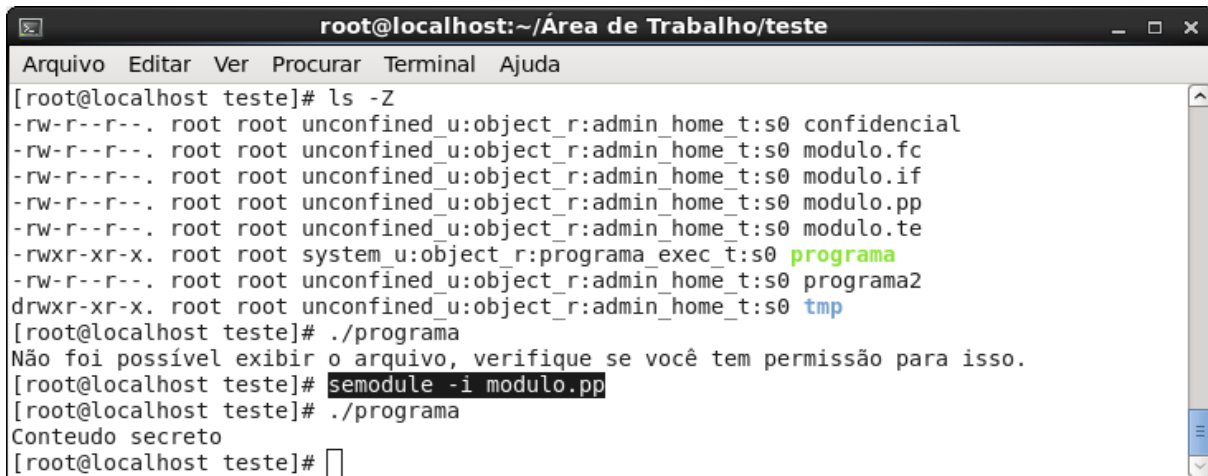


```
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.te
-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
[root@localhost teste]# make -f /usr/share/selinux/devel/Makefile
Compiling targeted modulo module
/usr/bin/checkmodule: loading policy configuration from tmp/modulo.tmp
/usr/bin/checkmodule: policy configuration loaded
/usr/bin/checkmodule: writing binary representation (version 10) to tmp/modulo.mod
Creating targeted modulo.pp policy package
rm tmp/modulo.mod.fc tmp/modulo.mod
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.fc
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.if
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.pp
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.te
-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
drwxr-xr-x. root root unconfined_u:object_r:admin_home_t:s0 tmp
```

Fonte: o próprio autor.

O módulo então compilado, pode ser instalado no SELinux através do comando “semanage --install <caminho do módulo>”. Assim, suas regras passam a funcionar no mesmo instante, conforme mostra a Figura 24.

Figura 24 - instalação de módulo com semodule.



```

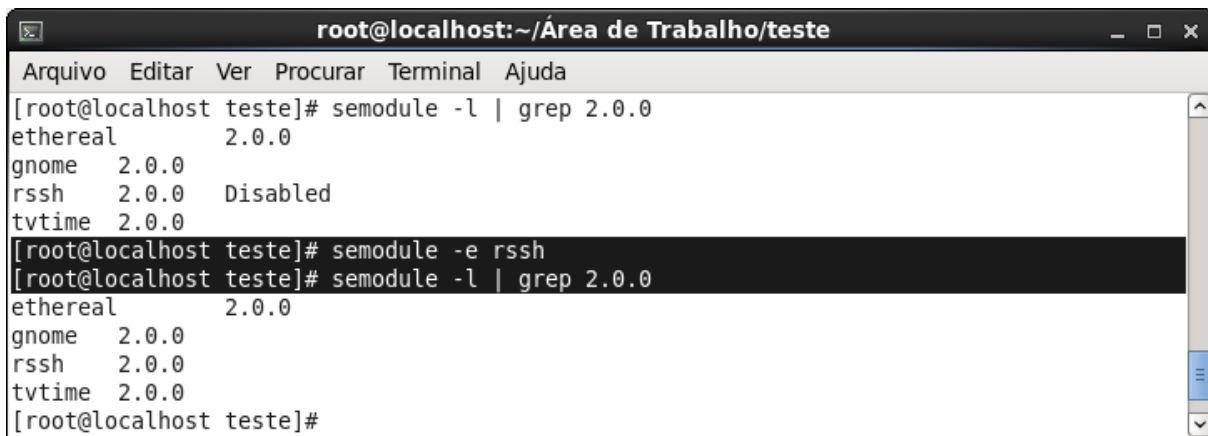
root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# ls -Z
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidencial
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.fc
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.if
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.pp
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 modulo.te
-rwxr-xr-x. root root system_u:object_r:programa_exec_t:s0 programa
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 programa2
drwxr-xr-x. root root unconfined_u:object_r:admin_home_t:s0 tmp
[root@localhost teste]# ./programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[root@localhost teste]# semodule -i modulo.pp
[root@localhost teste]# ./programa
Conteudo secreto
[root@localhost teste]#

```

Fonte: o próprio autor.

Para visualizar os módulos instalados e seu estado (ativos ou inativos), pode-se utilizar o comando “semodule -l”. Para ativar um módulo desativado utiliza-se o comando “semodule -e”, vide Figura 25:

Figura 25 - visualização de estado dos módulos e ativação com semodule.



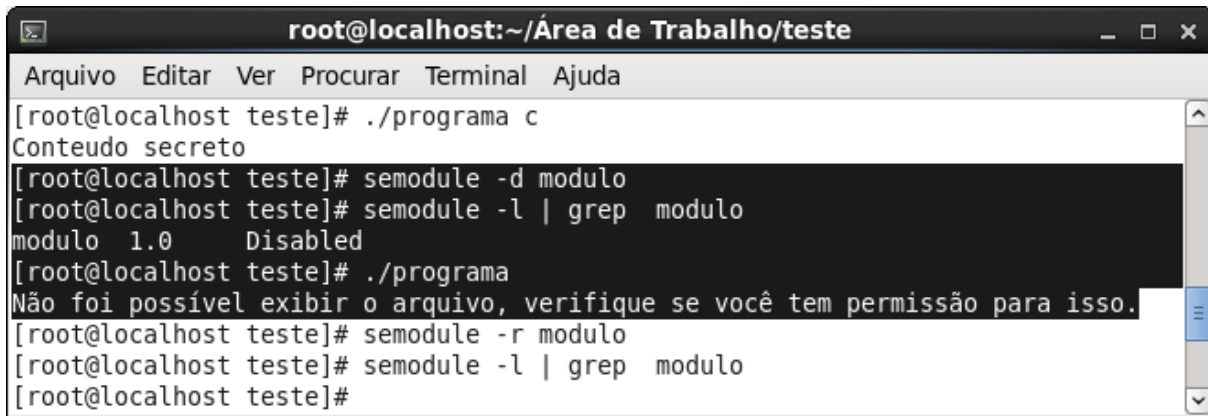
```

root@localhost:~/Área de Trabalho/teste
Arquivo Editar Ver Procurar Terminal Ajuda
[root@localhost teste]# semodule -l | grep 2.0.0
ethereal      2.0.0
gnome        2.0.0
rssh         2.0.0 Disabled
tvtime       2.0.0
[root@localhost teste]# semodule -e rssh
[root@localhost teste]# semodule -l | grep 2.0.0
ethereal      2.0.0
gnome        2.0.0
rssh         2.0.0
tvtime       2.0.0
[root@localhost teste]#

```

Fonte: o próprio autor.

Caso um módulo forneça uma permissão indesejada a um determinado processo, conforme vê-se na Figura 26, é possível desativá-lo, ou até mesmo removê-lo através dos comandos “semodule --disable <nome do módulo>” e “semodule --remove <nome do módulo>”, respectivamente.

Figura 26 - desativação e deleção de módulo com semodule.A terminal window titled "root@localhost:~/Área de Trabalho/teste" with a menu bar containing "Arquivo", "Editar", "Ver", "Procurar", "Terminal", and "Ajuda". The terminal output shows the following commands and their results:

```
[root@localhost teste]# ./programa c
Conteudo secreto
[root@localhost teste]# semodule -d modulo
[root@localhost teste]# semodule -l | grep modulo
modulo 1.0      Disabled
[root@localhost teste]# ./programa
Não foi possível exibir o arquivo, verifique se você tem permissão para isso.
[root@localhost teste]# semodule -r modulo
[root@localhost teste]# semodule -l | grep modulo
[root@localhost teste]#
```

Fonte: o próprio autor.

6.3 Cenário 3: Eficácia do SELinux

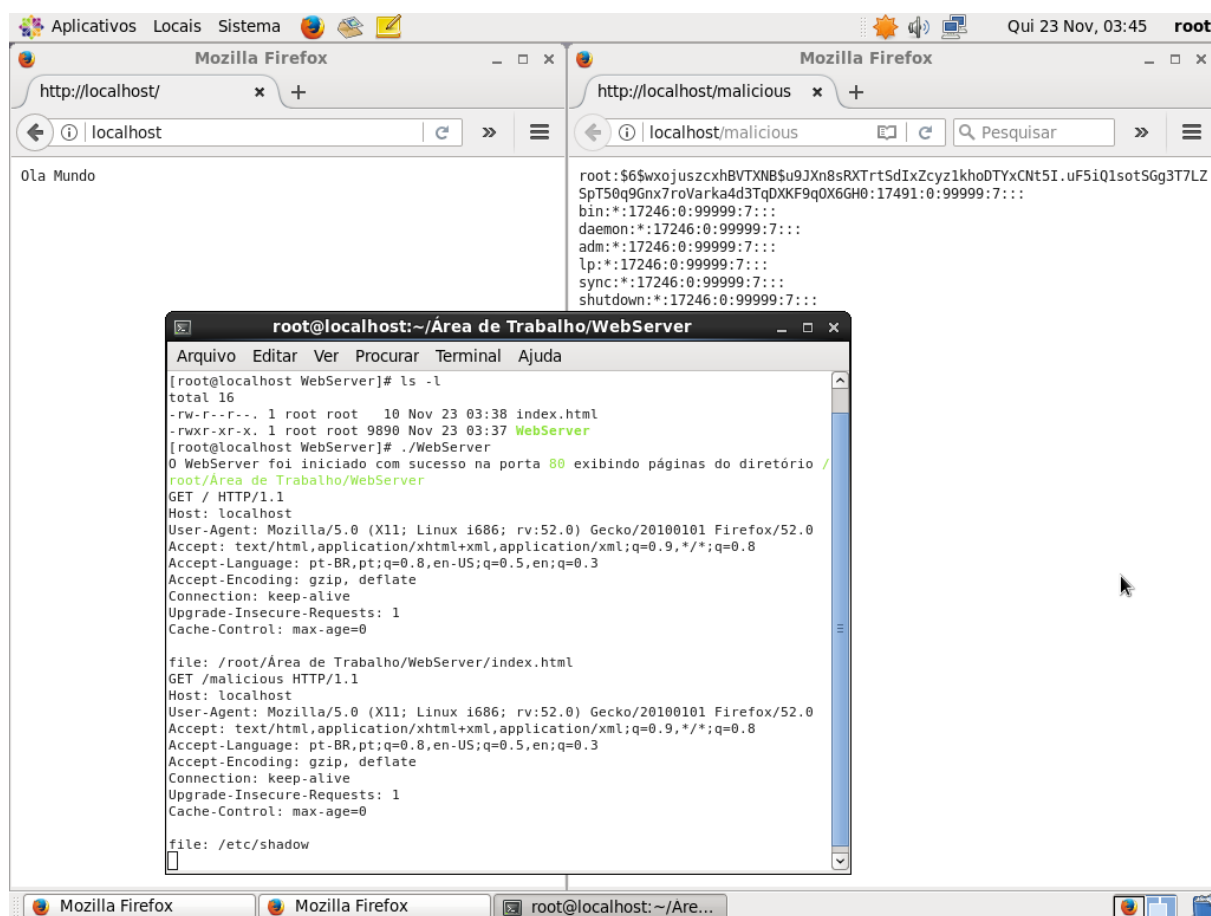
Um administrador de rede sem muitos conhecimentos em segurança, a fim de reduzir a quantidade de recursos de máquina consumidas por algumas aplicações web presentes na empresa, instala um *WebServer* - neste caso uma versão modificada do *WebServer* desenvolvido por Rastogi (2010) - de um repositório não oficial, cujo qual oferece a promessa de disponibilizar sites e aplicações *web* com alto desempenho e sem consumir muitos recursos.

Porém, este *WebServer* possui uma função maliciosa em seu código que permite ao seu desenvolvedor - que na verdade é um *hacker* - através do acesso a um determinado endereço dentro deste servidor, obter o conteúdo do arquivo “/etc/shadow”, tendo assim acesso ilegítimo às informações de login da máquina onde o servidor está em execução.

Antes de instalar tal *software* o administrador sequer checa os arquivos de código fonte e tal função passa despercebida. Para agravar a situação, o administrador instala o *software* e define que o usuário root irá executá-lo, concedendo assim acesso efetivo ao arquivo em questão para o desenvolvedor do *WebServer*, vide Figura 27.

O cenário retratado até então - onde o *hacker* consegue acessar o arquivo “/etc/shadow” - só foi viabilizado porque o DAC é o único mecanismo de controle de acesso na máquina. Como o usuário root é quem está executando o processo do *WebServer*, este processo pode manipular qualquer recurso da máquina, inclusive o arquivo em questão.

Figura 27 - WebServer com código malicioso exibindo “/etc/shadow” para seu criador.

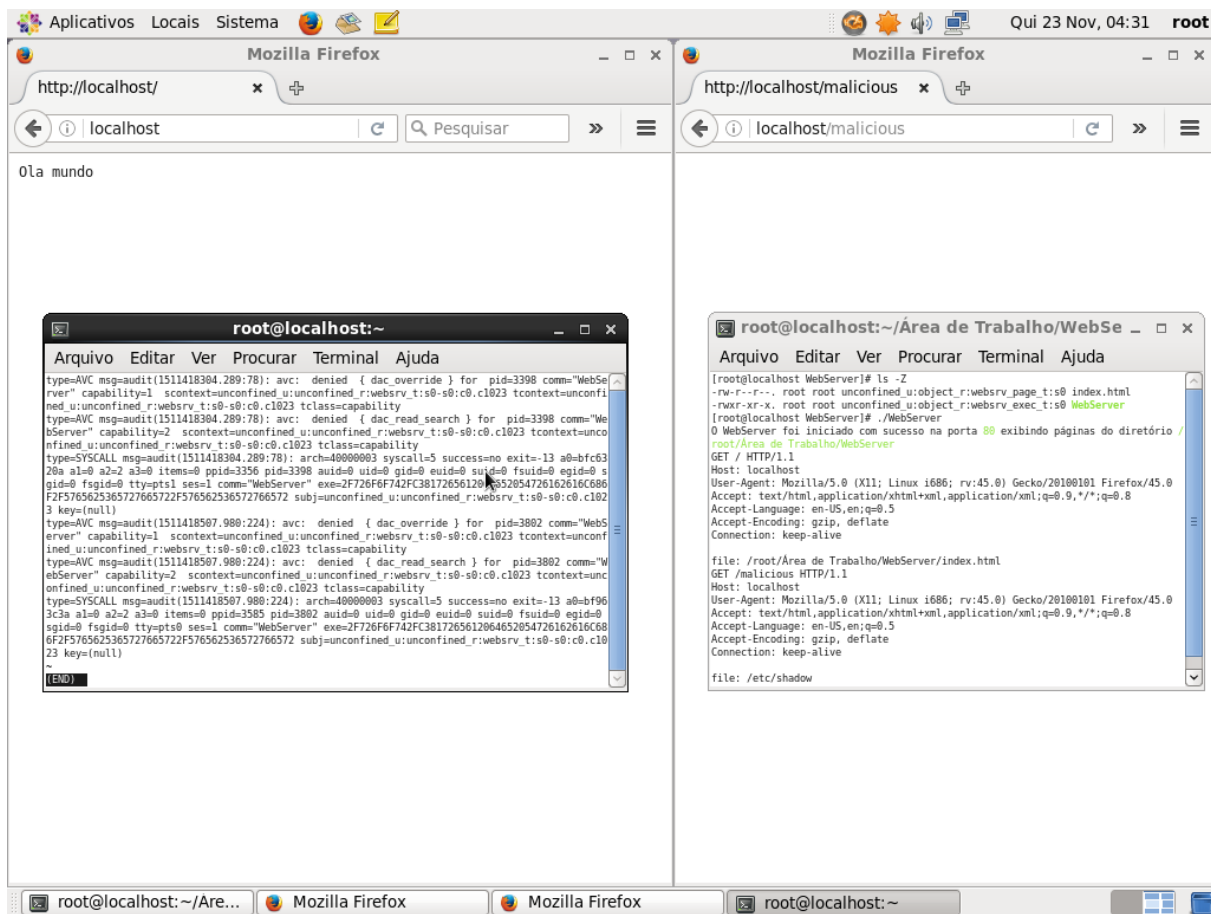


Fonte: o próprio autor.

Em outro cenário, o administrador instala o mesmo *software*, também executando-o como usuário root, porém com o SELinux ativo na máquina e a presença de um módulo de política para o *software*, conferindo a ele permissão apenas para realizar ações legítimas.

Neste cenário quando *hacker* realiza a tentativa de acesso ao arquivo “/etc/shadow”, nada é exibido em seu navegador e além disso o SELinux cria um alerta no arquivo de logs, permitindo ao administrador do sistema tomar conhecimento desta ação, conforme vê-se na Figura 28:

Figura 28 - SELinux impedindo a ação do hacker.



Fonte: o próprio autor.

É interessante notar na Figura 29, que um dos alertas gerados pelo SELinux relata o bloqueio do conjunto de *capabilities* “CAP_DAC_OVERRIDE” - conforme abordado em 3.4 – que é responsável por impedir que o usuário root tenha privilégios de acesso a arquivos restritos pelo DAC. É possível notar também outro alerta referindo-se a um conjunto de *capabilities* chamado “CAP_DAC_READ_SEARCH”, que segundo The Linux man-pages project (2017), teria a mesma função que o “CAP_DAC_OVERRIDE” (retratada em 3.4), exceto pelo fato de não possuir a autorização para modificar arquivos.

7 CONSIDERAÇÕES FINAIS

Segurança da Informação é uma área de extrema importância nas organizações. É possível atribuir o funcionamento pleno da organização à segurança de seus dados. Portanto é extremamente necessário que sejam aplicados procedimentos e métodos de segurança a fim de impedir o comprometimento desses dados.

Atualmente com a vasta utilização da tecnologia e a substituição da utilização dos papéis pela utilização de sistemas de TI, é necessário garantir a segurança desses sistemas, estendendo a eles a aplicação de procedimentos e métodos de segurança. Havendo a necessidade de proteção em TI, a preocupação com controle de acesso nos sistemas operacionais deve ser o primeiro fator a ser considerado.

Como já dito por Araújo (2017), não existe *software* totalmente seguro. Apesar disso, existem meios de endurecer a segurança. Devido à eficácia da aplicação de módulos MAC - como o SELinux, este modelo de controle de acesso torna-se muito útil para proteger informações.

Assim, conclui-se que acordando com os objetivos deste estudo, a utilização de um sistema computacional de segurança baseada em MAC – que é viabilizada pela LSM e seus módulos, como o SELinux - pode trazer enormes benefícios a indivíduos e organizações, trazendo à TI maior aproximação com os princípios e pilares de Segurança Informação, mantendo assim maior grau de segurança aos sistemas computacionais.

Considerando que o escopo deste trabalho aborda apenas conceitos básicos do SELinux e explora apenas a política *Targeted*, torna-se interessante, em trabalhos futuros, explorar outras formas de controle de acesso com o SELinux. Além disso, também torna-se interessante a exploração de outros módulos LSM, como AppArmor e Tomoyo.

REFERÊNCIAS

- ANNE, Surendra. **What is SUID and how to set SUID in Linux/Unix**, 2011. Disponível em: <<https://www.linuxnix.com/suid-set-suid-linuxunix/>>. Acesso em: 11 nov. 2017.
- APPARMOR. **AppArmor history**, 2012. Disponível em: <http://wiki.apparmor.net/index.php/AppArmor_History>. Acesso em: 16 nov. 2017.
- ARAUJO, Fábio. **Existe site 100% seguro?**, 2017. Disponível em: <<https://canaltech.com.br/seguranca/existe-site-100-seguro-100134/>>. Acesso em: 13 nov. 2017.
- BR DATA SECURITY. **Política de segurança da informação**, 2004. Disponível em: <<http://www.brdatanet.com.br/solucoes/politica.htm>>. Acesso em: 13 nov. 2017.
- CASSA, Monica. **A importância e a implementação da segurança da informação no âmbito das atividades de negócios**, 2003. Disponível em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/221>. Acesso em: 15 mar. 2017.
- CERT.BR. **Estatísticas dos incidentes reportados ao CERT.br**, 2017. Disponível em: <<https://www.cert.br/stats/incidentes/>>. Acesso em: 13 dez. 2017.
- COSTA, Ednílson. **Gestão em segurança: controle de acesso**. 2012. Disponível em: <<https://www.portaleducacao.com.br/conteudo/artigos/administracao/gestao-em-seguranca-controle-de-acesso/16115>>. Acesso em: 02 mar. 2017.
- COWAN, Crispin. **Linux security module Interface** [lista de distribuição]. Mensagem recebida por <linux-security-module-request@wirex.com>. Recebido em: 11 abr. 2001.
- ESCRITÓRIO DE GOVERNANÇA DE TI DO TRIBUNAL REGIONAL DA 11ª REGIÃO. **Comitê de segurança da informação**, 2011. Disponível em: <<http://governanca.trt11.jus.br/estrutura-de-ti/comite-de-seguranca-da-informacao-2/>>. Acesso em: 13 nov. 2017.
- JONES, M. **Anatomia do Security-Enhanced Linux (SELinux)**, 2008. Disponível em: <<https://www.ibm.com/developerworks/br/library/l-selinux/index.html>>. Acesso em: 13 nov. 2017.
- KOLLING, Rebeca. **Segurança da informação**. 2010. Disponível em: <<http://seguranca-da-informacao.info/>>. Acesso em: 15 mar. 2017.
- KOPROWIEC, Diogo; GEUS, Paulo. **Paradigmas da segurança em sistemas operacionais**. Disponível em: <<http://www.lasca.ic.unicamp.br/paulo/papers/2004-WSeg-diogo.kropiwiec-sistemas.operacionais.pdf>>. Acesso em: 02 jun. 2017.

MILLER, Todd. **Sudo in a Nutshell**, [s.d]. Disponível em: <<https://www.sudo.ws/intro.html>>. Acesso em: 11 nov. 2017.

NEMETH, Evi et al. **Manual completo do Linux: guia do administrador**. 2. São Paulo: Pearson, 2012.

NSA. **Contributors to SELinux**, 2016. Disponível em: <<https://www.nsa.gov/what-we-do/research/selinux/contributors.shtml>>. Acesso em: 16 nov. 2017.

O GLOBO. **Investimento em segurança da informação cresce mais no país**, 2015. Disponível em: <<https://oglobo.globo.com/economia/negocios/investimento-em-seguranca-da-informacao-cresce-mais-no-pais-17645471>>. Acesso em: 16 nov. 2017.

OLIVEIRA, Paulo Cesar. **Política de segurança da informação: definição, importância, elaboração e implementação**, 2013. Disponível em: <<https://www.profissionais.com.br/2013/06/politica-de-seguranca-da-informacao-definicao-importancia-elaboracao-e-implementacao/>>. Acesso em: 13 nov. 2017.

PROFISSÃO HACKER. **Pentest: Os testes de intrusão**, 2017. Disponível em: <<http://profissaohacker.com/servicos-profissionais/>>. Acesso em: 11 nov. 2017.

RASTOGI, Abhijeet. **A very simple http server written in C**, 2010. Disponível em: <<https://blog.abhi.host/blog/2010/04/15/very-simple-http-server-written-in-c/>>. Acesso em: 23 nov. 2017.

ROCHA, Rodrigo. **Pentest: O que é e porque é importante?**, 2013. Disponível em: <<http://www.rmaues.com/2013/06/pentest-o-que-e-e-porque-e-importante.html>>. Acesso em: 16 nov. 2017.

ROUSE, Margaret. **Mandatory access control (MAC)**, 2013. Disponível em: <<http://searchsecurity.techtarget.com/definition/mandatory-access-control-MAC>>. Acesso em: 16 nov. 2017.

SANTOS, Bruno Rocha. **Qual a importância do controle de acesso aos dados da minha empresa?**, 2016. Disponível em: <<http://inforrede.com.br/qual-a-importancia-do-controle-de-acesso-aos-dados-da-minha-empresa/>>. Acesso em: 02 mar. 2017.

SMALLEY, Stephen; FRASER, Timothy; VANCE, Chris. **Linux security modules: general security hooks for Linux**, [s.d.]. Disponível em: <<http://www.hep.by/gnu/kernel/lsm/>>. Acesso em: 16 nov. 2017.

THE CENTOS PROJECT. **How tos/SELinux**, 2017. Disponível em: <<https://wiki.centos.org/HowTos/SELinux>>. Acesso em: 20 nov. 2017.

THE LINUX FOUNDATION. **Overview Linux Kernel security features**, 2013. Disponível em: <<https://www.linux.com/learn/overview-linux-kernel-security-features>>. Acesso em : 13 nov. 2017.

THE LINUX MAN-PAGES PROJECT. **Su(1)**, 2014. Disponível em:
<<http://man7.org/linux/man-pages/man1/su.1.html>>. Acesso em: 24 nov. 2017.

THE LINUX MAN-PAGES PROJECT. **Capabilities(7)**, 2017. Disponível em:
<<http://man7.org/linux/man-pages/man7/capabilities.7.html>>. Acesso em: 24 nov. 2017.

TOMOYO. **A security module for system analysis and protection**, 2017. Disponível em: <<http://tomoyo.osdn.jp/>>. Acesso em: 16 nov. 2017.

ZANIQUELLI, Tiago. **Convergência segurança física e lógica**, 2010. Disponível em:
<<https://www.devmedia.com.br/convergencia-seguranca-fisica-e-logica/15760>>. Acesso em: 16 nov. 2017.

APÊNDICE A - Configuração, funcionamento e sintaxe do Sudo

A figura 31 ilustra parte do conteúdo do arquivo “/etc/sudoers”. Esta parte é responsável pela autorização de usuários do sistema a utilizarem o Sudo.

Figura 31 – Arquivo “/etc/sudoers”.

```

gusta@GPArchTP:/home/gusta
Arquivo Editar Ver Pesquisar Terminal Ajuda
GNU nano 2.8.7 Arquivo: /etc/sudoers Modificado

##
## Runas alias specification
##

##
## User privilege specification
##
root ALL=(ALL:ALL) ALL

## Uncomment to allow members of group wheel to execute any command
# %wheel ALL=(ALL:ALL) ALL

## Same thing without a password
# %wheel ALL=(ALL:ALL) NOPASSWD: ALL

## Uncomment to allow members of group sudo to execute any command
# %sudo ALL=(ALL:ALL) ALL

^G Ajuda      ^O Gravar     ^W Onde está? ^K Recort txt  ^J Justificar ^C Pos atual
^X Sair       ^R Ler o arq ^\ Substituir ^U Colar txt  ^T VerfOrtog ^_ Ir p/ linha

```

Fonte: O próprio autor.

É possível observar a existência da linha “root *ALL=(ALL:ALL) ALL*”. Para permitir a um usuário utilizar as funcionalidades do Sudo, o arquivo “/etc/sudoers” adota o formato “Usuário *Host*(Usuários:Grupos) Comandos”. Os campos deste formato possuem os seguintes atributos :

- **Usuário:** Usuário que poderá usar o Sudo, qual as regras definidas nos outros campos desta linha. (“*ALL*” indica que todos os usuários da máquina serão capazes de usar o Sudo);
- **Host:** Máquinas remotas cujas quais o usuário especificado pode utilizar o Sudo. (“*ALL*” indica que o usuário pode utilizar o Sudo para executar comandos nessa máquina conectando-se de qualquer máquina da rede);

- **Usuários:** Usuários cujos quais o usuário anteriormente especificado poderá utilizar do Sudo para executar comandos assumindo seus privilégios. (“ALL” indica que o usuário pode assumir os privilégios de qualquer usuário da máquina);
- **Grupos:** Grupos cujos quais o usuário poderá assumir privilégios para executar comandos, como se fosse um membro. (“ALL” indica que o usuário pode assumir os privilégios de qualquer grupo do sistema);
- **Comandos:** Comandos cujos quais o usuário pode executar enquanto utiliza as funcionalidades do Sudo. (“ALL” significa que o usuário pode executar todos os comandos possíveis no sistema);

Há também a linha “#%sudo ALL=(ALL:ALL) ALL”. Por padrão esta linha é comentada, com o caractere “#”, portanto não é considerada válida. O símbolo “%” no começo do campo “Usuário”, indica que ao invés de tratar-se de um usuário, sudo trata-se de um grupo e portanto, caso seja retirado o caractere “#” do início da linha, tornando-a válida, qualquer usuário pertencente ao grupo sudo será capaz de utilizar o Sudo de acordo com as regras especificadas nessa linha.

Na Figura 32, é importante notar que o grupo sudo teve que ser criado (groupadd), pois por padrão ele não existe no sistema. Após a criação do grupo, o usuário gusta foi adicionado a ele (gpasswd -a gusta sudo), conforme visto através do comando “id”.

Ao tentar listar o conteúdo do diretório pessoal do usuário root (ls /root) não se obteve sucesso, porém ao utilizar o Sudo para isso (sudo ls /root), a tentativa foi bem sucedida, isto porque o comando sudo quando utilizado sem uma opção “-u” ou “-g” definida, por padrão, assume os privilégios de usuário root e grupo root.

O mesmo ocorre com o usuário teste, porém é possível notar a utilização da opção “-u teste”. Esta opção faz com que os privilégios assumidos sejam os do usuário teste, assim é possível visualizar o conteúdo do diretório pessoal do usuário teste (sudo -u teste ls /home/teste).

Figura 32 – Exemplo de utilização do Sudo.

```

gusta@GPArchTP:~
Arquivo Editar Ver Pesquisar Terminal Ajuda
[root@GPArchTP gusta]# groupadd sudo
[root@GPArchTP gusta]# gpasswd -a gusta sudo
Adicionando usuário gusta ao grupo sudo
[root@GPArchTP gusta]# su gusta
[gusta@GPArchTP ~]$ id
uid=1000(gusta) gid=1000(gusta) grupos=1000(gusta),56(bumblebee),1003(sudo)
[gusta@GPArchTP ~]$ ls /root/
ls: não foi possível abrir o diretório '/root/': Permissão negada
[gusta@GPArchTP ~]$ sudo ls /root/
'Área de trabalho'  Downloads  Modelos  Público
Documentos         Imagens   Música   Vídeos
[gusta@GPArchTP ~]$ ls /home/teste/
ls: não foi possível abrir o diretório '/home/teste/': Permissão negada
[gusta@GPArchTP ~]$ sudo -u teste ls /home/teste/
Desktop Documents Music Pictures Videos
[gusta@GPArchTP ~]$ sudo -u teste ls /home/gusta
ls: não foi possível abrir o diretório '/home/gusta': Permissão negada
[gusta@GPArchTP ~]$ ls /home/gusta/
'Área de trabalho'  Modelos      Público
Documentos         Música       q
Downloads          NetBeansProjects 'systemctl disable teclado'
go                 pgadmin.log  Vídeos
Imagens            Projects     'VirtualBox VMs'
[gusta@GPArchTP ~]$ ls -l /home
total 24
drwx----- 32 gusta gusta 4096 nov 12 01:09 gusta
drwx-----  2 root  root 16384 ago 26 22:59 lost+found
drwx-----  7 teste teste 4096 nov 12 01:11 teste
[gusta@GPArchTP ~]$

```

Fonte: O próprio autor.

Porém, ao tentar visualizar o conteúdo do diretório pessoal do usuário gusta com a opção “-u teste”, a tentativa foi frustrada (`sudo -u teste /home/gusta`), e ao realizar a mesma tentativa sem o sudo (`ls /home/gusta`), a visualização foi possível. Isto ocorre porque ao utilizar o Sudo, os privilégios do usuário que está executando o comando são totalmente ignorados, e os que passam a ser válidos são apenas os adotados pelo Sudo - root se as opções “-u” e “-g” não forem utilizadas, ou usuário/grupo definidos nessas opções – e como o usuário teste não possui permissão de leitura para o diretório pessoal do usuário gusta (`ls -l /home`), não foi possível listar seu conteúdo.

APÊNDICE B - Sintaxe do comando "su"

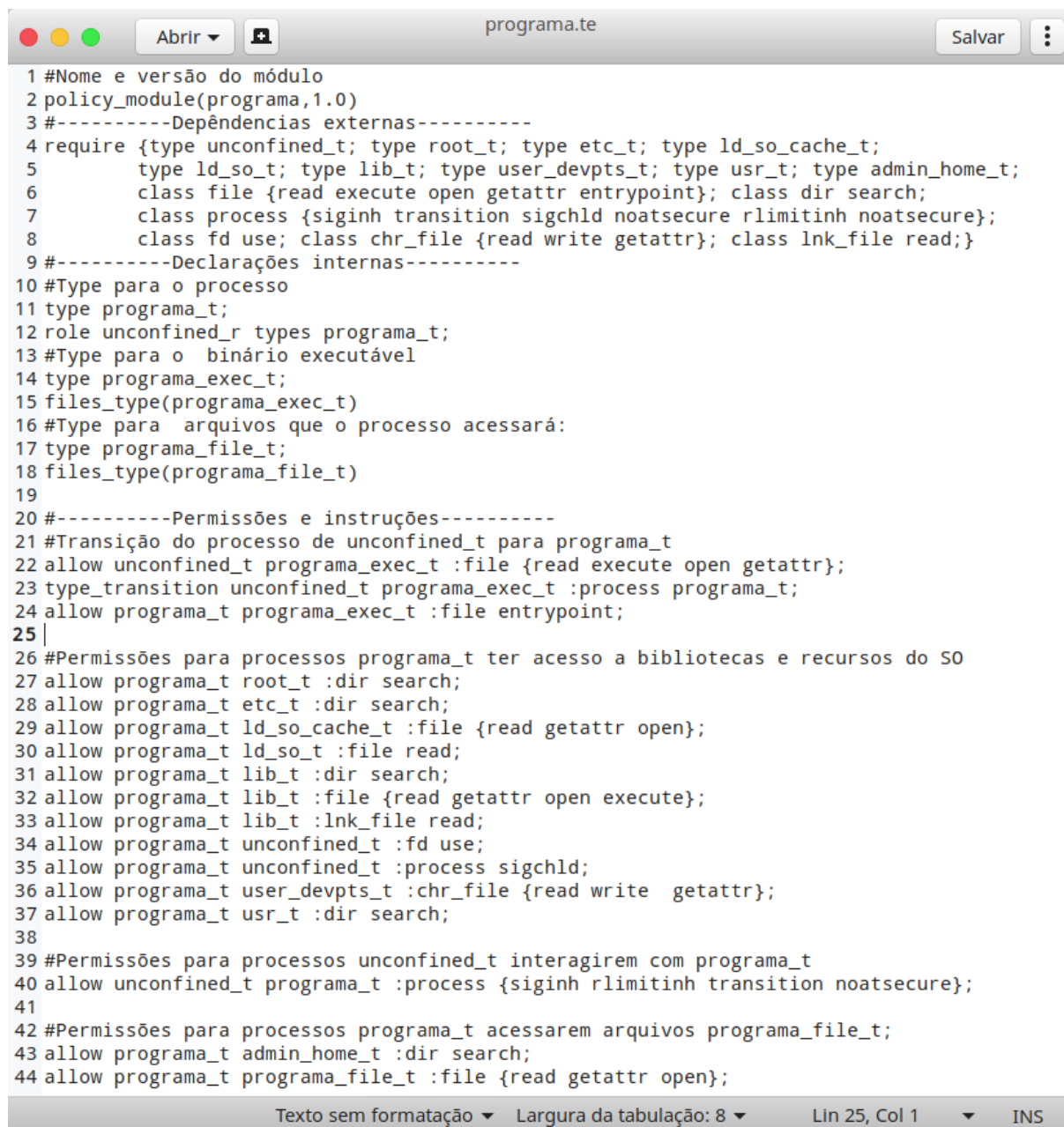
O comando possui a seguinte sintaxe: "su <opções> <-> <usuário>". Esta sintaxe prevê a utilização de um "-" entre as opções e o usuário, porém este caractere trata-se de um alias à opção "--login", que deve ser utilizada sempre que outra opção for utilizada, porém, se nenhuma outra opção for utilizada, esta opção pode ser suprimida. Assim, se o usuário que deseja-se instanciar o *shell* é o usuário *gusta*, utiliza-se o comando "su *gusta*". Além disso, não é necessário informar o usuário caso deseje-se instanciar o *shell* do root, utilizando o comando apenas em sua forma mais básica, "su" (The Linux man-pages project, 2014).

APÊNDICE C - Processo de atribuição e visualização do SUID

Para definir o SUID, a forma mais prática é utilizar o comando “chmod u+s <arquivo>”, onde “chmod” é o nome do comando utilizado para gerenciar permissões no sistema, “u” define que as permissões a serem alteradas são as pertinentes ao usuário proprietário, “+s” atribui o SUID, e “<arquivo>” trata-se do caminho do arquivo cujo qual terá as permissões alteradas (ANNE, 2011).

APÊNDICE D - Estrutura de um módulo SELinux

Figura 33 - Exemplo de módulo SELinux.



```

1 #Nome e versão do módulo
2 policy_module(programa,1.0)
3 #-----Depêndencias externas-----
4 require {type unconfined_t; type root_t; type etc_t; type ld_so_cache_t;
5         type ld_so_t; type lib_t; type user_devpts_t; type usr_t; type admin_home_t;
6         class file {read execute open getattr entrypoint}; class dir search;
7         class process {siginh transition sigchld noatsecure rlimitinh noatsecure};
8         class fd use; class chr_file {read write getattr}; class lnk_file read;}
9 #-----Declarações internas-----
10 #Type para o processo
11 type programa_t;
12 role unconfined_r types programa_t;
13 #Type para o binário executável
14 type programa_exec_t;
15 files_type(programa_exec_t)
16 #Type para arquivos que o processo acessará:
17 type programa_file_t;
18 files_type(programa_file_t)
19
20 #-----Permissões e instruções-----
21 #Transição do processo de unconfined_t para programa_t
22 allow unconfined_t programa_exec_t :file {read execute open getattr};
23 type_transition unconfined_t programa_exec_t :process programa_t;
24 allow programa_t programa_exec_t :file entrypoint;
25 |
26 #Permissões para processos programa_t ter acesso a bibliotecas e recursos do SO
27 allow programa_t root_t :dir search;
28 allow programa_t etc_t :dir search;
29 allow programa_t ld_so_cache_t :file {read getattr open};
30 allow programa_t ld_so_t :file read;
31 allow programa_t lib_t :dir search;
32 allow programa_t lib_t :file {read getattr open execute};
33 allow programa_t lib_t :lnk_file read;
34 allow programa_t unconfined_t :fd use;
35 allow programa_t unconfined_t :process sigchld;
36 allow programa_t user_devpts_t :chr_file {read write getattr};
37 allow programa_t usr_t :dir search;
38
39 #Permissões para processos unconfined_t interagirem com programa_t
40 allow unconfined_t programa_t :process {siginh rlimitinh transition noatsecure};
41
42 #Permissões para processos programa_t acessarem arquivos programa_file_t;
43 allow programa_t admin_home_t :dir search;
44 allow programa_t programa_file_t :file {read getattr open};

```

Texto sem formatação ▾ Largura da tabulação: 8 ▾ Lin 25, Col 1 ▾ INS

Fonte: O próprio autor.

A Figura 33 retrata um exemplo de módulo SELinux. Este módulo foi desenvolvido para implementar na política *Targeted* a possibilidade de restringir a ação do programa retratado na Figura 5 (programa cuja função é ler o conteúdo de um arquivo com o nome “confidencial”).

Um módulo de política SELinux é composto basicamente de quatro seções. Como é possível ver no exemplo da Figura 33, são estas seções:

- Nome e versão do módulo: o nome e a versão do módulo são “programa” e “1.0”, respectivamente. A fim de identificar a qual software o módulo refere-se, convém dar ao módulo o mesmo nome do software.
- Dependências externas: para definir regras de acesso, é necessário estabelecer relações com contextos criados por outros módulos. Assim, faz-se necessário declarar os contextos externos quais são referenciados neste módulo. O mesmo ocorre com funções do SO (Sistema Operacional).
- Declarações internas: esta seção é responsável por declarar e inserir os contextos no SELinux, de modo que seja possível atribuí-los aos objetos do sistema.
- Permissões e instruções: esta seção é responsável por definir as regras de interação entre os contextos declarados nas seções anteriores (segunda e na terceira) e, definir instruções que serão executadas pelo SELinux - como por exemplo definir a transição de contexto de um processo, permitindo que o processo deixe de ser executado com o mesmo contexto do processo pai (no caso do *bash*, o *shell* padrão do CentOS, o contexto possui o atributo *Type* igual a “*unconfined_t*”) e passe a ser executado com o contexto definido pelo módulo (neste caso, “*programa_t*”).

É importante notar, que na seção de permissões e instruções, estão contidas permissões de acesso a recursos do sistema operacional e do sistema de arquivos. Isto ocorre porque quando o SELinux está ativo em modo restritivo, ocorre confinamento destes recursos, e como o programa depende de interagir com eles para funcionar, é necessário estabelecer relações de permissão com estes recursos.

A partir da linha 42 no exemplo da Figura 33 estão as permissões de interação entre os objetos com contexto “*programa_t*” e “*programa_file_t*”. Em outras palavras, estas linhas estabelecem que um processo com contexto “*programa_t*” pode pesquisar arquivos em um diretório com contexto “*admin_home_t*” (pasta pessoal do usuário root, visto que o programa em questão encontra-se em um subdiretório desta pasta) e

manipular arquivos (permissão para ler, obter dados sobre o arquivo e abrir) contidos em diretórios com o contexto “programa_file_t”.

As regras seguem o formato: “<permissão> <contexto do objeto requerente> <contexto do objeto manipulado> :<classe requerida do objeto manipulado> <{funções da classe requerida, separadas por espaço};>”.

Utilizando a linha 43 – que permite a processos do contexto “programa_t” buscar arquivos dentro do diretório “/root”, a regra toma o seguinte formato: “allow programa_t admin_home_t :dir search;”.

É possível notar, que o caractere “;” está presente ao final de todas as regras e, que o caractere que define a separação entre o fim de um campo e o começo de outro é um espaço em branco.

Outro ponto importante é que quando o campo “<{funções da classe requerida, separadas por espaço}>” referir-se a apenas uma função, torna-se dispensável o uso dos caracteres “{” e “}” no começo e no fim do campo, respectivamente.