CENTRO UNIVERSITÁRIO - CESMAC

FUNDAÇÃO EDUCACIONAL JAYME DE ALTAVILA - FEJAL

CURSO DE PÓS-GRADUAÇÃO "LATU SENSU"

REDE DE COMPUTADORES

Um estudo sobre o uso de Controle de Acesso Pró-Ativo Baseado em Papéis para a Contenção de Falhas de Segurança Desconhecidas em Serviços no Ambiente GNU/Linux.

Leandro Inácio Santos de Carvalho

Maceió - AL

2011

Leandro Inácio Santos de Carvalho

Um estudo sobre o uso de Controle de Acesso Pró-Ativo Baseado em Papéis para a Contenção de Falhas de Segurança Desconhecidas em Serviços no Ambiente GNU/Linux.

Maceió - AL
2011

LEANDRO INÁCIO SANTOS DE CARVALHO

Um estudo sobre o uso de Controle de Acesso Pró-Ativo Baseado em Papéis para a Contenção de Falhas de Segurança Desconhecidas em Serviços no Ambiente GNU/Linux.

Artigo apresentado como requisito final para obtenção do título de Especialista no Curso de Pós-Graduação em Redes de Computadores, pelo Professor Mestre Breno Jacinto Duarte da Costa, da Faculdade de Ciências Exatas e Tecnológicas, Centro de Estudo de Maceió - CESMAC.

Maceió - AL
2011

Leandro Inácio Santos de Carvalho

Um estudo sobre o uso de Controle de Acesso Pró-Ativo Baseado em Papéis para a Contenção de Falhas de Segurança Desconhecidas em Serviços no Ambiente GNU/Linux.

Aprovado em 24 de Setembro de 2011

_____

Prof. Me. Breno Jacinto Duarte da Costa – Orientador

Maceió – AL
2011

A todos que de alguma forma puderam contribuir para a realização de mais um sonho.

Agradecimentos

Agradeço a minha esposa por todo o apoio dado e paciência, apesar de todas as dificuldades. A minha Mãe, por ter dado suporte em todas as minhas escolhas, ao meu Tio que da mesma forma o fez, meus Irmãos, meu Pai, por ser meu guia, minha motivação, minha paixão e pela educação dada.

A todos que de uma forma ou de outra me ensinaram a viver, a escutar e a praticar o que aprendi. Pelos conhecimentos adquiridos nesses quase dois anos de pós-graduação, aos amigos de trabalho e os amigos do projeto Arch Linux Brasil, que incentivaram a iniciativa deste trabalho, tornando realidade e uma contribuição para todos os usuários desta distribuição que amo e uso, Arch Linux.

Ao meu orientador Breno Jacinto pela paciência em ter esperado esse trabalho sair da mente ir para o papel.

“Ter falhado significa ter lutado.
Ter lutado significa ter crescido.”
Maltbie Badcok

UM ESTUDO SOBRE O USO DE CONTROLE DE ACESSO PRÓ-ATIVO BASEADO EM PAPÉIS PARA A CONTENÇÃO DE FALHAS DE SEGURANÇA DESCONHECIDAS EM SERVIÇOS NO AMBIENTE GNU/LINUX.

Leandro Inácio Santos de Carvalho
Breno Jacinto Duarte da Costa (Orientador)

RESUMO: A busca por Controle de Acesso é um requisito fundamental na Segurança da Informação. Neste Trabalho, realizamos um estudo aprofundado sobre os principais mecanismos de controle de acesso existentes, dando uma ênfase maior ao modelo de controle de acesso baseado em papéis (RBAC). É realizado um estudo de caso onde avaliamos as limitações do modelo de acesso discricionário (DAC) em ambientes Linux, assim como também é avaliada uma ferramenta que implementa RBAC como um conjunto de remendos (*patches*) para o Kernel do Linux. O estudo de caso mostra um experimento onde o controle de acesso pode melhorar a segurança do Sistema Operacional como um todo, sendo possível conter ataques a vulnerabilidades desconhecidas.

PALAVRAS-CHAVE: RBAC; grsecurity; fortalecimendo; Arch Linux; Segurança.

ABSTRACT: The quest for Access Control is a fundamental requirement in Information Security. In this work, we conducted a thorough study of the major mechanisms of existing access control mechanisms, giving greater emphasis to the role-based access control model (RBAC). We conducted a case study where the limitations of the discretionary access model (DAC) are evaluated in a Linux test-bed. A tool implementing RBAC as a set of patches for the Linux kernel is also evaluated. The case study shows an experiment where access control can improve the security of the operating system as a whole, showing that it is possible to contain attacks of unknown vulnerabilities.

KEYWORDS: RBAC; grsecurity; *hardening*; Arch Linux; Security.

INTRODUÇÃO

Dentro do contexto dos Sistemas Operacionais (SO) da atualidade, a questão do Controle de Acesso é um requisito básico. A evolução dos sistemas *UNIX-like* (hoje representados na figura do OpenBSD, FreeBSD, NetBSD e Linux, no mundo do Código Aberto), o controle de acesso já era um requisito obrigatório já que o UNIX foi, desde o início, um SO multi-usuário (RAYMOND, 2003). Ambientes da Microsoft, baseados no SO Windows, foram sistemas monousuários, onde se pode dizer que não havia controle de acesso

de forma alguma. Até a versão do Windows 98, o SO não apresentava sequer uma forma de autenticação de usuários: todos tinham privilégio máximo.

A Internet fez com que os fabricantes e desenvolvedores de SO passassem a implementar formas de controle de acesso dos recursos do SO aos usuários, levando em conta uma realidade multitarefa, multiusuário e de compartilhamento de recursos entre os processos – memória e CPU. Tais características incrementaram consideravelmente a usabilidade dos SO modernos, no entanto, gerou grande complexidade para isolar os recursos sendo compartilhados pelos processos. Não durou muito até que a integridade dos dados sendo compartilhados em memória pudesse ser comprometida, gerando resultados que variavam de uma simples instabilidade ao completo controle remoto da CPU em softwares provendo algum tipo de serviço de Internet, como os servidores Web.

De acordo com o site do grupo CERT.br (Grupo de Resposta a Incidentes de Segurança para a Internet brasileira) no meses de Abril a Junho, "notificações sobre ataques a servidores Web cresceram 14% em relação ao trimestre anterior e 69% em relação ao mesmo período de 2010" (Figura 1).

Tabela: Totais Mensais e Trimestral Classificados por Tipo de Ataque.

Mês	Total	worm (%)		dos (%)		invasão (%)		web (%)		**scan** (%)		**fraude** (%)		outros (%)	
abr	**44908**	1157	2	10	0	12	0	901	2	7269	16	3891	8	31668	70
mai	**42715**	1038	2	2	0	5	0	1091	2	6628	15	3307	7	30644	71
jun	**39458**	933	2	2	0	2	0	1183	3	7950	20	2544	6	26844	68
Total	**127081**	3128	2	14	0	19	0	3175	2	21847	17	9742	7	89156	70

Figura 1 Estatística de Ataques mais Comuns em 2010

A exposição dos serviços de rede na Internet, aliado ao entendimento incompleto dos problemas no nível do SO e do software provendo o serviço na rede, levou a ondas de ataques que fizeram história na Internet. A Figura 1 mostra que, recentemente, ainda há uma incidência alta de computadores infectados por *Malware (*vírus, *worms*, cavalos de tróia, *spyware).*

As melhores práticas dentro da Administração de Sistemas pregam que o administrador deve estabelecer, em sua política de segurança, as atualizações periódicas de segurança no SO e nas aplicações, análise de logs, *firewalls*, Sistemas de Detecção por Intruso (SDI), dentre outras práticas como descrito na cartilha de segurança proposto pelo CERT.br[1].

[1]Esta cartilha de segurança pode ser encontrada no seguinte endereço: http://cartilha.cert.br/.

Embora tais práticas sejam importantes, algumas ressalvas devem ser feitas. Primeiro, nem sempre é possível manter o SO e os aplicativos atualizados de todas as falhas, somente daquelas que já são conhecidas e já existem correções (*patches*) disponíveis. Segundo, *Firewalls* podem atuar de forma muito eficiente no perímetro da Rede e até mesmo em máquinas individuais (os chamados "*Firewalls* Pessoais"). Falhas de segurança conhecidas como "*Zero Day*", aquela onde não se conhece, publicamente, a causa, nem o efeito e que, portanto não há correção para ela, não podem ser contidas com as práticas mencionadas. Isso só é possível hoje através do Controle de Acesso Pró-Ativo.

Por Pró-Ativo queremos dizer que não é necessário conhecer as vulnerabilidades que venham a afetar um determinado serviço, uma vez que o estabelecimento de políticas de segurança bem estruturadas será capaz de conter ataques que visam explorar tais vulnerabilidades no nível da interação dos processos e das chamadas de sistemas disponibilizadas pelo próprio SO aos aplicativos.

Utilizando técnicas como *sandboxing* que consiste no isolamento ou confinamento da execução de uma aplicação a possíveis danos que possam resultar da exploração bem-sucedida de uma vulnerabilidade. Este confinamento é realizado de maneira granular, gerenciando memória e acesso a recursos do sistema pelo aplicativo confinado. Além disso, estudaremos também mecanismos que bloqueiam genericamente falhas de segurança em nível de software, como bugs de *Buffer Overflows*[2] e *Format Strings*[3].

Neste trabalho, é feito um estudo sobre o Controle de Acesso em Sistemas Operacionais, fazendo um levantamento desde os modelos conceituais, às implementações que tem sido utilizada pela maioria dos SO da atualidade. Especificamente, é enfatizado o modelo de controle de acesso RBAC (*Role Based Access Control* – Controle de Acesso Baseado em Papéis) e como ele pode ser aplicado no nível de *kernel* do SO de maneira Pró-Ativa para conter falhas de segurança ainda desconhecidas.

Desta forma, o artigo está dividido da seguinte forma. Na seção 1 são apresentadas as bases teóricas sobre Controle de Acesso em geral. Na seção 2, discutimos brevemente sobre a metodologia de pesquisa usada neste trabalho. O estudo de caso é apresentado na seção 3, e os resultados obtidos do experimento na seção 4.

1 FUNDAMENTAÇÃO TEÓRICA

[2]*Buffer overflow* conforme MATEUS (2011) é "um tipo de erro associado à escrita em regiões de memória sem a devida precaução de se verificar seus limites".
[3]*Format Strings* conforme YAMAZAKI (2009) é um ataque que "permite ao atacante escrever ou ler de qualquer ponto da memória mapedada de um processo".

Em um Sistema Operacional moderno, multiusuário e multitarefa, há componentes que protegem o espaço alocado na memória, CPU e outros recursos para não serem tomados por um único processo e até mesmo por aqueles que não tenham autorização de uso desses recursos. Esta abordagem é importante para que o compartilhamento de recursos seja feito de forma segura e justa, usando um espaço lógico comum. Para isso é necessário que sejam estabelecidas Políticas de Acesso no SO.

> "Acesso é a habilidade de fazer algo com algum recurso computacional (ex. usar, modificar e ver). Controle de acesso é o significado pelo qual a habilidade está explicitamente habilitada ou restrita de alguma forma (usualmente através do meio físico e dos controles baseados em sistemas). O controle de acesso baseado em computador pode prescrever não apenas quem pode ou o que pode ter acesso em um recurso específico do sistema, mas também o tipo de acesso que é permitido. Estes controles podem ser implementados no sistema computacional ou um dispositivo externo" (NIST/ITL 1995).

Desta forma, os objetivos iniciais do Controle de Acesso são a manutenção da integridade do SO e a proteção de todos os usuários do sistema, através da autenticação, autorização e auditoria. Assim, é possível, por exemplo, identificar o usuário na hora de acesso ao sistema, dando as devidas permissões ou tarefas que ele poderá realizar (o que está ou não autorizado a fazer) no sistema, mantendo registros das ações do usuário (*logs*).

MATTOS (2003) traz o conceito de objeto e sujeito. O objeto "representa o recurso computacional cujo acesso é controlado". Cada objeto tem um tipo de acesso diferente. Assim, os objetos são abstrações fornecidas pelo sistema que podem ou não ter acesso a outros objetos, operações e recursos previamente autorizadas. A definição de sujeito é citada abaixo:

> "Define-se sujeito (*subject*) como sendo a representação do usuário dentro do sistema. Pode-se entender o conceito de sujeito como o de um processo no sistema aliado a um conjunto de credenciais de acesso que associam aquele processo a um usuário da base do sistema. Em geral, as regras de autorização são expressas utilizando-se o identificador do usuário (seu login ou identificação única no sistema). No momento da autenticação, as credenciais de acesso são geradas para aquela sessão do usuário. A partir daí, qualquer execução de processo ou rotina dentro do sistema é encarada como um *subject*, já que alia as credenciais de acesso do usuário com um processo que o representa no sistema." (MATTOS, 2003).

O controle de acesso especifica os tipos de operações e o conjunto de objetos ao qual o sujeito terá acesso. Cada objeto possui um identificador único dentro do sistema e são acessados apenas através de operações realizadas pelos sujeitos donos dos objetos, definindo assim o direito de acesso. Os acessos pode ser representados de diversas formas (as implementações variam), no entanto uma convenção simples adotada é: leitura (*read)*, representado pela letra r, escrita (*write)*, representada pela letra w, execução (*eXecute)* pela letra x.

De forma abstrata pode-se visualizar o modelo de proteção como uma matriz, sendo as linhas os sujeitos e as colunas os objetos, contendo os direitos de acessos em cada entrada. "A partir de uma matriz de acesso desdobram-se naturalmente listas de controle de acesso e lista de competência (*capabilities*), técnicas utilizadas na maior parte dos sistemas operacionais modernos" (RA01, citado por MATTOS, 2003). Conforme exemplificado na Tabela 1 que se segue.

Sujeito/Objeto	arquivo1	arquivo2	arquivo3
Luke	leitura (r), escrita (w)	-	escrita (w)
Skywalker	-	execução (x), leitura (r), escrita (w)	-
Darth	-	leitura (r)	leitura (r)
Vader	leitura (r)	-	-

Tabela 1 exemplo de matriz de acesso

No exemplo da Tabela 1, mostra-se a convenção para sistemas *UNIX-like*, onde é utilizada a matriz de acesso para autorização simples, sendo inviável sua utilização em uma grande quantidade de usuário por objetos.

Seu uso pode ser simplificado utilizando lista de controle de acesso (ACLs), onde a matriz pode ser indexada pela coluna, indicando o usuário e quais as suas permissões em relação ao objeto. A ACL é armazenada junto com o objeto já relacionando as permissões de cada usuário no sistema. Conforme LENTO (2006) *et al* isso "facilita determinar quais os modos de acesso que os sujeitos estão autorizados em um objeto, provendo uma forma fácil de rever ou revogar os modos de acessos aos objetos".

Através da matriz de acesso cada sujeito está relacionado a uma lista de competência (*capability*), ou seja, quais as permissões de acesso o sujeito possui (neste caso, as linha da matriz). LENTO (2006) *et al* afirma que as listas de competência "permitem fácil verificação e revogação dos acessos autorizados para um determinado sujeito".

Isto significa que cada *capability* corresponde a um identificador protegido, identificando e especificando cada objeto e os direitos de acesso atribuídos ao sujeito (proprietário daquele objeto), tem duas características fundamentais, a *capability* pode passar de um sujeito para outro podendo assim modificá-la, sem a necessidade de consultar o sistema para realizar tal tarefa.

As ACLs são extensamente utilizadas em SOs populares da atualidade, como o Linux e Windows 7, sendo usada no nível mais alto do sistema (nível de usuário), dentro do modelo do modelo DAC (*Discretionary Access Control*), descrito na próxima seção.

1.1 O Modelo de Controle de Acesso Discricionário (DAC)

O modelo DAC (*Discretionary Access Control* - Controle de Acesso Discricionário) é largamente utilizado atualmente, desde aplicações comerciais, a sistemas operacionais e banco de dados relacionais.

Sua definição inicial surgiu no livro do Departamento de Defesa dos EUA chamado *Trusted Computer System Evaluation Criteria* (NSA 1985), conhecido como *Orange Book*. Atualmente o mesmo é substituído pela norma ISO/IEC 15408-1:2009, também chamado de *Common Criteria*.

O *Orange Book* diz que um sistema discricionário precisa fornecer os requisitos básicos de segurança por meio da separação entre os usuários e os dados. Conforme o NSA (1985), o *Trusted Computing Base* (TCB) define e controla o acesso entre sujeitos (usuários) e objetos (e.g, arquivos e programas) num sistema ADP (*Automatic Data Processing*). O mecanismo de aplicação (e.g, usuário, grupo, controles públicos, lista de controle de acesso) deve permitir aos usuários especificar e controlar o compartilhamento desses objetos por usuários indicados, grupos ou ambos.

No modelo DAC o *owner* (dono) de um objeto pode conceder a outros usuários a permissão de acessá-lo em um modo qualquer de operação, podendo a qualquer momento ser revogada ou não essa permissão.

De acordo com OSBORN *et al.* (2000), a ideia central do DAC é a de que o proprietário de um objeto, que normalmente é o seu criador, tem poder discricionário sobre quem mais pode acessar o objeto. Desta forma, é permitido aos administradores e aos outros usuários (sujeitos) especificarem e controlarem os recursos com outros usuários.

O modo como o modelo DAC define seus usuários, dividindo-o em categorias como administrador, super-usuário (*root*) e não administrador, este último chamando-o de usuários

comuns. Esta abordagem apresenta alguns problemas, por exemplo: se, eventualmente, for necessário à execução de algum serviço que precise de privilégio um pouco mais elevado, será necessário total privilégio de administrador. Tentando contornar tal problema e tornar sua execução um pouco mais segura, usa-se o princípio do menor privilégio, onde se "preza por delegar somente os privilégios necessários para que um determinado objeto possa realizar sua função na organização. Tomando o cuidado de não atribuir privilégios menores que o necessário, pois, desta forma, tal objeto não conseguirá realizar suas tarefas" (PASSOS 2004).

Essa abordagem do menor privilégio acaba sendo muito permissiva. Por exemplo, consideremos um servidor Web. É possível que o processo em execução seja levado a acessar, de forma indevida, algum arquivo fora de seu ambiente de execução ou até mesmo executar códigos arbitrários, com o intuito de elevar ainda mais o privilégio e conseguir acesso de superusuário no SO.

Um exemplo prático pode esclarecer como o DAC é implementado em um ambiente real. Em um SO como Linux, é possível observar o DAC em ação no cenário abaixo:

```
$ ls -l script.sh
-rwxrw-r-- 1 leandro users 0 Aug 27 14:27 script.sh
```

Observe que a execução do comando mostrou uma linha iniciando com 10 (dez) caracteres, onde separa-se o primeiro caracter e indica ser um arquivo comum representado pelo sinal (-), podendo ele ter as seguintes definições a de um diretório (d), arquivo especial de caracter (c), arquivo especial de bloco (b) ou um link (l). Os outros 9 (nove) são divididos em 3 (três) partes de 3 (três) caracteres definindo assim as permissões do usuário como leitura (r), gravação/escrita (w) e execução (x) como "rwx", do grupo (users) como "rw-" apenas leitura e escrita e os "outros" como "r--", apenas como leitura.

Em ambientes militares, onde é necessário um alto nível de controle e resistência a diversos tipos de ataques, o modelo DAC é muito permissivo. Este foi o principal motivador para a elaboração e criação do modelo MAC, descrito na seção seguinte.

1.2 O Modelo de Controle de Acesso Mandatório (MAC)

O modelo DAC, apesar de simples implementação e gerenciamento, apresenta limitações. O modelo MAC (*Mandatory Access Control*), ao contrário do modelo anterior que tinha um foco nos ambientes comercial e educacional, está direcionado a ambientes militares e governamentais.

Da mesma forma que o modelo DAC, encontramos a descrição do modelo MAC no *Orange Book - Trusted Computer System Evaluation Criteria* (NSA 1985).

NSA (1985) define como deve ser aplicada uma política de controle de acesso de todos os sujeitos e objetos sob seu controle. Neste modelo os sujeitos e objetos são atribuídos a rótulos de segurança, sendo uma combinação nos níveis de classificação hierárquica e não hierárquica das categorias de segurança, os rótulos são usados como base nas decisões do controle de acesso. De acordo com a TCB (*Trusted Computing Base*), deve ser capaz de suportar um ou mais níveis de segurança.

No modelo MAC, um sujeito só pode ler um objeto se seu nível de segurança for maior ou igual à classificação hierárquica de segurança do objeto e as categorias não-hierárquica no nível de segurança do sujeito incluem todas as categorias não-hierárquica em nível de segurança do objeto. Em relação à escrita, o sujeito só pode escrever no objeto se a classificação hierárquica em nível de segurança for menor ou igual à classificação de segurança do objeto e todas as categorias não-hierárquicas no nível de acesso estão inclusos no nível de segurança do objeto.

Os dados de identificação e autenticação devem ser utilizados para identificar e garantir o acesso e a segurança do sujeito, com base em rótulos que dependem das definições dos atributos estendidos (*xattrs*) aplicados a cada objeto, como pode ser visto na **Figura 2**.

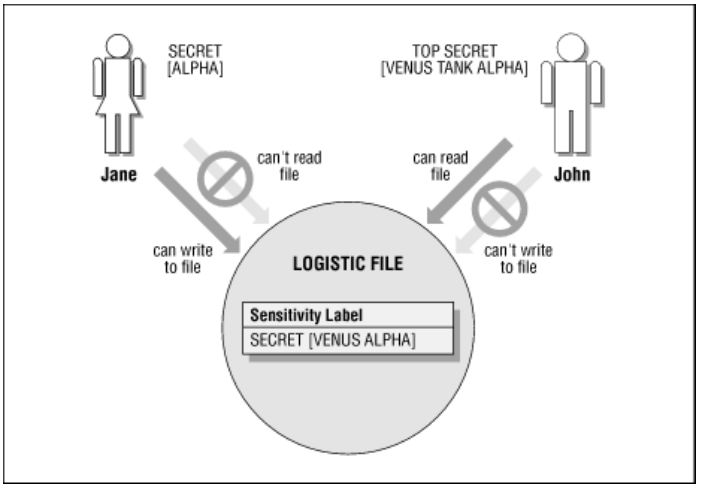

Figura 2 Níveis de acesso com base em rótulos

Portanto, a política de controle de acesso mandatório é expressa em termos de rótulos de segurança ligado a sujeitos e objetos. Um rótulo em um objeto é chamado de classificação de segurança, enquanto um rótulo em um usuário é chamado, certificado de segurança

(OSBORN, 2000), ou seja, uma maneira de restringir o acesso a um objeto é baseado no nível de segurança da informação contida neste objeto.

Conforme SANDHU (1993) citado por OSBORN (2000), o modelo clássico MAC é construído especificamente para incorporar a política de fluxo unidirecional de informações em uma estrutura. Este fluxo de informação unidirecional pode ser aplicado para a confidencialidade, integridade, confidencialidade e integridade em conjunto, ou de políticas de agregação, tais como a Muralha da China.

Por ser considerado mais efetivo do que o DAC, o MAC prevê que o sujeito não pode realizar escolhas de quem tem ou não acesso aos seus objetos, assim os objetos não podem ser sobrescritos pelos seus donos. Desta forma, os sujeitos não podem dar acesso a outros que não são autorizados, conforme as políticas definidas pelo sistema.

As permissões de acesso não são fornecidas pelo dono do objeto, mas sim pelo próprio sistema. As políticas do MAC são implementadas pelo sistema e não pode ser aplicado, modificado ou removido, se não por uma operação privilegiada, em nível mais baixo (*kernel* do SO) ou por um usuário que possuir um papel específico. As regras de segurança aplicadas pelo sistema precisam referir-se aos objetos do sistema e que não são protegidos pelo DAC. Se comparados de forma simples, pode-se considerar como lista branca de acesso o MAC e lista negra de acesso o DAC.

Portanto, não adianta ser dono do objeto se o sistema interfere diretamente no acesso daquele objeto. Desta forma, as permissões dadas pelo modelo DAC são úteis apenas para definir quais usuários podem ou não acessar aquele arquivo. Ao aplicar o MAC para o controle de acesso de um software aos recursos do SO, isso ocorre na forma de uma "caixa de areia" (*sandbox*) que limitam a capacidade de atuação do software.

As implementações do modelo MAC hoje podem ser encontradas na forma do SELinux (um dos pioneiros), Smack, AppArmor e TOMOYO.

1.3 Controle de Acesso Baseado em Pápeis (RBAC)

Enquanto o modelo DAC mostrou-se muito permissivo, o modelo MAC mostrou-se muito proibitivo em muitos cenários. Buscando um equilíbrio, o modelo RBAC (*Roled Based Access Control* - Controle de Acesso Baseado em Pápeis) foi desenvolvido. Disponível em poucos sistemas, "sua utilização é apropriada para sistemas que processam informações sensíveis mas não classificadas, bem como para aqueles sistemas que processam informações classificadas" (NSA/ITL, 1995).

O modelo RBAC, diferentemente dos modelos já citados, visa controlar o acesso através de papéis exercidos por cada sujeito dentro do sistema. O conceito básico do RBAC é: usuários são atribuídos a funções e as permissões são atribuídas a papéis e para os usuários conseguirem permissões, precisam ser membros daqueles papéis. NSA/ITL (1995) completa que, "o processo de definição dos papéis deve ser baseada em uma análise profunda de como uma organização opera e deve incluir a entrada de um amplo espectro de usuários em uma organização". A análise dos papéis para a criação das políticas de controle de acesso, precisam incluir possíveis mudanças e evoluções dentro do sistema, com o intuito de evitar possíveis problemas de gerenciamento.

A proposta para o modelo RBAC, conhecido como "*Core* RBAC", apareceu por volta de 1992 em um artigo escrito por FERRAIOLO (1992) na 15th *National Computer Security Conference*, em Baltimore nos EUA, afirmando que "a confiança no DAC como principalmente método de controle de acesso é infundado e inadequado para muitas organizações governamentais, comerciais e civis". E ainda complementa que o RBAC é "um controle de acesso não discricionário e mais central para as necessidades de processamento seguro de e para sistemas não militares do que o DAC".

O modelo proposto por FERRAIOLO (1992) foi integrado ao quadro do SANDHU *et al* (2000) com a intenção de unificar e definir um padrão de implementação do modelo RBAC, assim, ficando conhecido como NIST RBAC, sendo adotado como padrão ANSI/INCITS em 2004, conforme o indicado no site do NIST (*National Institute of Standard and Technology*).

Conforme SANDHU (2000), o padrão NIST do modelo RBAC é "organizado em uma sequência de quatro etapas para aumentar da capacidade funcional, chamados de RBAC linear, RBAC hierárquico, RBAC de restrição e RBAC simétrico. Estes níveis são cumulativos e cada em sequência complementa os requisitos do anterior". O padrão NIST RBAC, requer que os papéis e as atribuições entre usuários e permissões sejam de muitos para muitos.

Como se trabalha com papéis, o conceito básico é simples, as permissões são estabelecidas dando direito de acesso através das funções exercidas e pelo uso dos recursos do sistema, sendo restrita para os sujeitos autorizados a assumir aquele papel ao invés de identificar o nível de segurança do usuário. As mesmas são eficazes para o desenvolvimento e aplicação de políticas específicas e menos custoso no processo de administração de segurança.

A forma de trabalho de cada usuário implica nas funções e restrições aos recursos do sistema, os papéis exercidos podem ser revogados ou acrescentados conforme as necessidades

surgirem na realização de sua função. Além disso, a associação de um usuário a um novo papel pode ser veiculado se novas atribuições surgirem dentro do sistema ou organização, similarmente a atribuições antigas que não fizerem mais parte ou não sejam mais necessárias para o uso e possam ser revogadas. Ou seja, os papéis podem representar tarefas, responsabilidades e qualificações associadas àquela função dentro do sistema.

Desta forma, as políticas de segurança podem mudar e evoluir. Uma das vantagens da utilização do RBAC é a simplicidade como o administrador ou o analista de segurança pode atualizar os controles de acesso sem atualizar os privilégios de cada usuário de forma individual. SANDHU (2000) afirma que o "RBAC fornece um poderoso mecanismo para reduzir a complexidade, custo e potencial de erro na atribuição de permissões do usuário dentro da empresa".

1.3.1 RBAC Linear

O RBAC linear é obrigatório em qualquer tipo de implementação do modelo RBAC e tem por exigência em sua implementação, a revisão do papel pelo qual as funções atribuídas a um usuário específico, bem como funções específicas atribuídas a vários usuários possam ser determinados. Isto significa que um usuário tenha várias permissões para exercer vários papéis, acabando com o problema de ter que permitir a execução de uma tarefa por vez.

Ficando implícito e diferenciando-o do conceito de grupos apresentado em outros modelos. "O modelo NIST RBAC reconhece o tradicional controle de acesso baseado em grupo como o primeiro nível do RBAC por ser amplamente implantado e ser uma tecnologia familiar servindo como ponto de partida para o RBAC" (SANDHU 2000). Outro ponto do RBAC linear, é que sua definição determina quais recursos ou não devem ser excluídos dos papéis. Cada implementação de forma independente pode especificar se existe ou não limitação na quantidade de papéis que um usuário pode assumir e quantos usuários podem ser associados a um papel, observando que deve existir escalabilidade neste ponto.

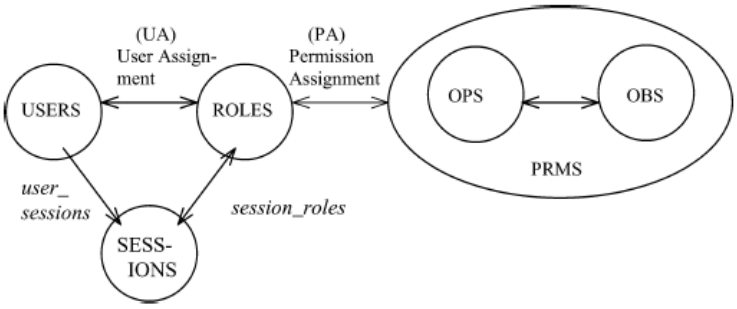

Figura 3 RBAC Linear

Algumas definições que ajudam a entender melhor o RBAC linear e dos que se seguem nas figuras utilizadas como a anterior, são listadas abaixo. "O conceito de sessão não é parte explicita do RBAC linear. Uma sessão corresponde a uma ocasião especial, quando um usuários faz login no sistema para realizar uma atividade" (SANDHU *et al.* 2000), conforme o padrão NIST RBAC os sistemas devem ser capazes de ativarem os papéis dos usuários durante o login em uma única sessão:

- U = *Users* (Usuários);
- R = *Roles* (Papéis);
- OPS = *Operations* (Operações);
- OBS = *Objects* (Objetos);
- user_sessions (Sessões do usuário);
- session_roles (permissões da sessão);
- *Sessions* (Sessão);
- P = PRMS = *Permissions* (Permissões);
- UA = *User Assignment* (Atribuição do Usuário);
- PA = *Permission Role Assignment* (Atribuição da Permissão ao Papel);
- RH = *Role Hierarchy* (Hierarquia de Papel);
- SSD = *Static Separation Of Duties* (Separação de funções estática);
- DSD = *Dynamic Separation of Duties* (Separação de funções dinâmica).

Esta ativação pode ser altamente restritiva se os papéis forem limitados pela quantidade de sessões, alguns sistemas ativam apenas um papel por sessão.

Um administrador pode criar e organizar os papéis em uma ordem hierárquica representando a função exercida pelo sujeito em um sistema ou organização. Os sujeitos são veiculados em um ou mais papéis, representando as funções exercidas por eles. É importante

salientar que apesar de um usuário estar veiculado a um papel, não quer dizer que ele esteja em outro papel para exercer a mesma função, para isto precisa-se da análise tentando evitar sobreposição.

Um dos aspectos de segurança já abordados anteriormente é, o conceito do menor privilégio, que requer a identificação minuciosa das atribuições para determinar o menor número de privilégios para realizar aquela função e restringir o sujeito ao domínio daquele papel. Consequentemente para alcançar um resultado adequado o controle sobre as análises e permissões precisam ser detalhados. Pois quanto menos permissões, mais segurança, sendo que a mesma pode implicar em dificuldades na realização dos trabalhos e na parte de gerenciamento das permissões que para contornar este inconveniente pode-se criar mais papéis e veicular o sujeito a eles, conforme surgirem as demandas.

Um dos inconvenientes de utilizar esta abordagem, é que determinadas atribuições acabam tendo responsabilidades que se sobrepõem a realidade daquela função o que causa a veiculação de sujeito a uma categoria que permita mais privilégios do que o necessário e acaba sendo penosa a forma de acesso sob medida, através de atributos ou restrições, causando acesso ilegal a algum recurso ou informação. Isso significa que pode existir sobreposições de atribuições, ou seja, um sujeito pode acabar precisando de um papel a mais e desnecessário para realizar tarefas comuns.

1.3.2 RBAC Hierárquico

Com o intuito de evitar estas sobreposições, ineficiência e trabalho repetitivo, são criadas tarefas comuns para todos os sujeitos do sistema além das específicas para cada função. Este inoportuno trabalho, faz surgir uma característica interessante deste modelo de controle de acesso, hierarquia de papel. A mesma pode ser estabelecida com o intuito de fornecer uma estrutura organizacional dentro do sistema. Esta hierarquia define os papéis de cada função contendo atributos únicos além de conter outros papéis, ou seja, onde um papel contém de forma implícita as atribuições que estão veiculadas a outro papel sem a necessidade de um administrador precisar informar explicitamente que um papel acima na hierarquia possa realizar as tarefas de um papel abaixo. Além disso, esta hierarquia é a forma natural de organizar as funções para reproduzir as responsabilidades, competências e autoridade de cada usuário dentro do sistema.

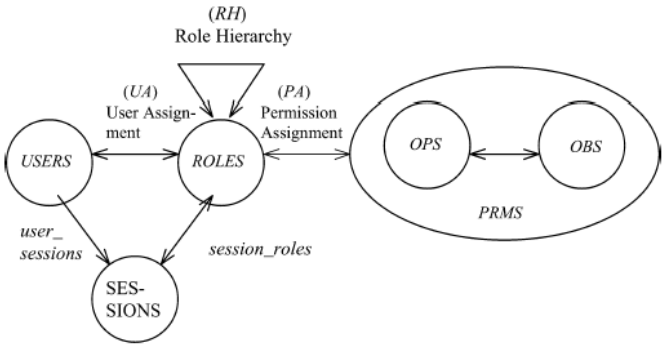

Figura 4 RBAC Hierárquico

O modelo NIST, divide a hierarquia em dois subníveis, geral e limitada. Na geral existe o suporte para servir como hierarquia em ordem arbitrária, já na limitada alguns sistemas impõem restrições, onde suas estruturas são limitadas para servir apenas como uma árvore ou árvore invertida.

Árvore invertida facilita o compartilhamento de recursos, mas não permite a agregação de novos recursos a partir de mais de um papel. Já a árvore normal, agrega permissões de outros papéis, mas não compartilha recursos.

Os recursos podem ser compartilhados através de herança, onde o acionamento desta implica na utilização do papel abaixo dela, consequentemente traz o inconveniente de ceder permissões além do necessário. Com a ativação de hierarquia, não existe problema em relação ao acionamento de uma hierarquia superior pois o mesmo é ativado durante a sessão do usuário, ou o compartilhamento podem utilizar ambos. SANDHU *et al.* (2000) afirma que "hierarquias de papel na forma de ordens arbitrárias parciais é a característica mais desejável, além de RBAC linear", sendo que "existem produtos que suportam apenas hierarquia limitada, substancialmente melhorando a capacidade além do modelo linear". A figura que se segue demonstra como funciona a hierarquia limitada, onde utiliza-se uma de árvore e árvore invertida (figura 5).

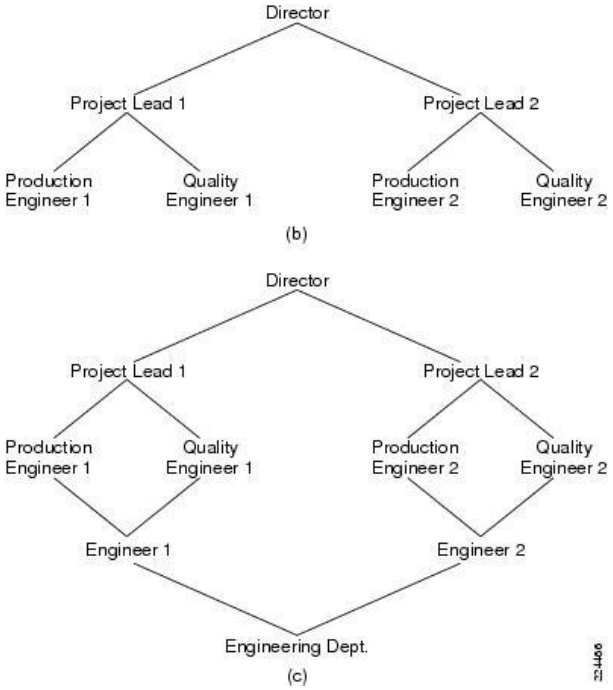

Figura 5 Árvore e Árvore Invertida

1.3.3 RBAC de Restrição

O RBAC de restrição impõe como requisito a separação das funções (conhecido como, SOD - *Separation Of Duties*), uma técnica para reduzir a possibilidade de fraudes e danos acidentais, espalhando a reponsabilidade e autoridade de uma tarefa por vários papéis (figura 6). Além de impedir o conflito nas políticas de acesso evitando que um papel adquira permissões de forma indevida.

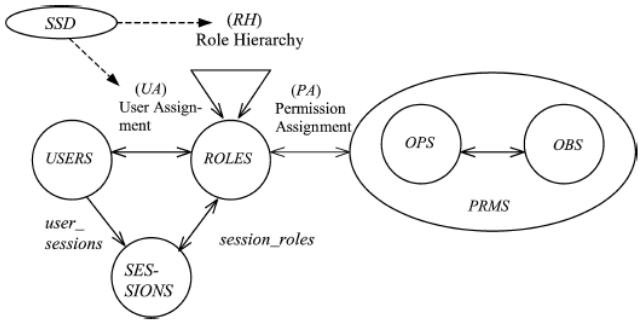

Figura 6 RBAC com divisão de atribuições estáticas (SSD)

Existe separação de funções estática (SSD - *Static Separation of Duties*), usando atribuição de papel (UA - *User Assignment*), pode ser centralmente específico e depois ser uniformemente imposto a um papel específico, e dinâmico (DSD - *Dynamic Separation of Duties*), ativação de papel na sessão do usuário, as permissões são concedidas quando os

papéis não constituem conflito e podem atuar independentemente, excluindo o papel anterior e menos permissivo para não assumir ambos ao mesmo tempo.

A utilização do RBAC de restrição em conjunto com RBAC hierárquico, por muitas vezes é considerado como uma instanciação do modelo MAC. RBAC de restrição são considerados herança no RBAC hierárquico. Restrições DSD (figura 7) podem ser hierarquicamente relacionados ao contrário da SSD.

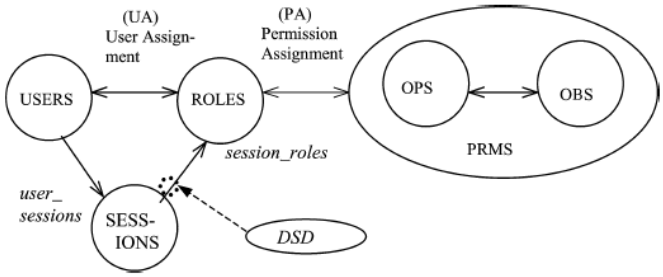

Figura 7 RBAC com divisão de atribuições dinâmicas (DSD)

O RBAC fornece uma unidade de controle que pode ser referenciada em regras individuais, para descrever os detalhes relevantes de segurança e restrições de cada usuário, a qual não consegue ser determinada em um modelo de controle de acesso simples. Além de fornecer aos administradores capacidade para dizer que ação deve ser realizada por um sujeito, quando, de onde, em qual ordem e/ou apenas tarefas relacionadas pode ser executadas. Estas definições podem ser limitadas, algumas regras podem ser copiadas para um determinado número de sujeitos ou um sujeito por vez e por tempo determinado.

1.3.4 RBAC Simétrico

O RBAC simétrico amplia o requisito de revisão do papel no RBAC linear, para definição de qual usuário pode assumir aquela função. O resultado desta revisão pode ser direta, pertence ao conjunto de permissões atribuídas ao usuário e/ou papel, e indireta, pertence ao conjunto de permissões incluídas na direta, além de as permissões que são atribuídos ao usuário. “Assim, os papéis para que uma permissão específica possa ser determinada, bem como permissões atribuídas a uma função específica. A performance da revisão da permissão deve ser efetivamente comparável à de revisão da função do usuário” (SANDHU 2000).

Apenas utilizado no último nível do modelo NIST, por ser difícil de implementar em larga escala como em sistemas distribuídos, onde as responsabilidades do administrador podem ser divididas entre os domínios de proteção central e local, ou seja, as políticas de proteção central podem ser definidas em um nível empresarial, deixando as questões de defesa que são de interesses locais, em nível de unidade organizacional. Este modelo diz ser intrínseco ao RBAC distinguindo-o dos controles de acesso baseado em grupo.

1.3.4 Vantagens do Modelo RBAC

Pode-se afirmar que o modelo RBAC, apresenta as seguintes vantagens:

- Permite aos usuários realizarem uma ampla quantidade de tarefas;
- Proporciona flexibilidade e uma considerável facilidade de aplicação;
- A partir de uma estrutura estática e dinâmica, regulam-se as tarefas dos usuários estabelecendo papéis, interações, restrições e hierarquia de papéis;
- Através da utilização de hierarquia os administradores podem abstrair de forma natural os controles de acesso;
- Definido os papéis, o administrador só revoga ou concede as permissões (papéis).

Tais vantagens entram em contraste com os modelos apresentados anteriormente, onde é oferecido um controle de acesso direto e geralmente menos intuitivo para o Administrador, já que os recursos disponíveis são as ACLs, *capabilities* e ações de objeto a objeto.

O RBAC pode ser usado para gerenciar privilégios em sistemas de qualquer porte, e, atualmente, seu uso é considerado como uma boa prática de segurança (SANDHU, 2000). Pode-se encontrar implementações do RBAC no *Active Directory* (AD) e SQL Server da Microsoft, nos SO FreeBSD, SELinux, Solaris, e nos SGBD Oracle e PostgreSQL. Em ambientes Linux é possível acrescentar suporte de RBAC em nível de kernel através do grSecurity, que é o estudo de caso deste trabalho.

2 METODOLOGIA DE PESQUISA

Toda a pesquisa até aqui utilizada e proposta, deu-se utilizando a Internet a partir de sites como Google, Google Acadêmico, *Scielo* e leitura de Artigos, Monografias, Dissertações

e Padrões para os modelos de controle de acesso, sempre com a finalidade de agregar conhecimento ao tema, para realização do trabalho, ora proposto.

A abordagem utilizada para elaboração do trabalho prático se deu pela necessidade de manter um ambiente seguro para uma organização que visa disponibilizar um serviço *online*, sempre analisando os aspectos de tempo e disponibilidade de correção da aplicação utilizada caso haja problemas de segurança ou falhas da aplicação, com intuito de evitar indisponibilidade do serviço e acessos indevidos a informações e ao sistema.

A parte prática do trabalho se dá utilizando a distribuição Arch Linux em um ambiente virtualizado, além da compilação em um ambiente *chroot*[4] do kernel com grsecurity no padrão de gerenciamento de pacotes da distribuição.

3 ESTUDO DE CASO

Nesta seção descreveremos nosso estudo de caso, começando por um estudo das limitações do modelo DAC dentro do ambiente Linux e a adição de algumas características visando combater tais limitações (seção 3.1). Na seção 3.2, apresentaremos o grsecurity, um conjunto de *patches* para o kernel do Linux que permite o controle de acesso baseado em papéis no nível de kernel.

3.1 Estudo da Implementação do modelo DAC no SO Linux

Como visto anteriormente, o modelo DAC apresenta uma falha através do princípio do privilégio mínimo. Desta forma no Linux, é implementado através das permissões especiais, *setuid*, *setgid* e *sticky*. Observe o comando *passwd*, que possui a permissão *setuid*, a qual permite um usuário executar o comando ou programa como se fosse o super usuário (root). A utilização do *setuid* faz com que todo usuário se transforme no usuário dono do comando em execução. Isto signifca que se um programa estiver com permissão de root e com o *setuid* (SUID) ativado, qualquer usuário ao executar este programa, terá privilégios de root.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 24704 Jun 26 04:12 /usr/bin/passwd
```

Nos caracteres que definem as permissões do usuário, fica evidente a utilização desta permissão “rws” onde no lugar de “x” (execução) nas permissões básicas temos “s”, caracterizando o *setuid*. Tão qual *setgid* (SGID) quando estiver ativado e executado por um

[4]É uma técnica que se assemelha ao *sandboxing.*

usuário, a execução acontece como se o grupo dono fosse o mesmo do arquivo, em diretórios, os arquivos criados tem como grupo dono o grupo do diretório com o SGID. Concluindo assim tem-se a permissão *sticky*:

```
$ ls -ld /tmp
drwxrwxrwt 19 root root 4096 Aug 27 14:36 /tmp
```

Seu uso em diretórios faz com que apenas o dono do arquivo e do diretório possa remover o arquivo, diretórios e modifiquem o conteúdo contido dentro dos arquivos. Utilizado apenas em diretórios temporários, onde qualquer usuário pode criar arquivos. Afirma-se ainda que antigamente quando aplicado em arquivos, informava ao SO para não utilizar memória RAM na execução deste arquivo.

Pode-se realizar o controle a partir de ACL's (*Access Control List* - Lista de Controle de Acesso), com o uso do comando *setfacl*, o mesmo define as ACL's para arquivos e diretórios e para visualizar as permissões utiliza o comando *getfacl*, da mesma forma que se trabalha para definir as permissões usando *chmod*, usado para definir as permissões e *chown*, usado para mudar usuário e grupo.

Um detalhe interesse é a evolução da implementação do nível de acesso no Linux, onde existe atualmente a implementação do PAM (*Plugglable Autentication Modules* - Módulos de Autenticação Plugáveis) para auxiliar no controle de recursos e usuários no sistema, definindo a forma como um programa autentica e aloca os recursos para o usuário sem a necessidade de recompilação por parte daquele aplicativo.

3.2 GrSecurity

Vários modelos de controle de acesso tem sido integrado ao Kernel do Linux de forma a incrementar o modelo DAC já presente por padrão. O SELinux[5], por exemplo, implementa o modelo MAC, sendo inicialmente desenvolvido pela NSA (*National Security Agency*) dos EUA, é integrado ao kernel a partir do framework LSM (*Linux Security Modules*).

O grsecurity[6] é um conjunto de *patches* aplicados diretamente ao kernel do Linux Kernel implementando o controle de acesso RBAC de maneira pró-ativa. Os desenvolvedores afimam que o "grsecurity é o único que fornece proteção contra *zero-day* e outras ameaças

[5] http://www.nsa.gov/research/selinux site do projeto contendo as informações relevantes sobre SELinux.

[6] http://grsecurity.net site do projeto contendo informações sobre grsecurity.

avançadas, que tomam o tempo do administrador para corrigir o problema nas distribuições em testes para aplicar em produção".

O grsecurity ainda adiciona uma camada de autenticação para os administradores, além de auditar eventos importantes no sistema isolando-o sem a necessidade de configuração manual através do RBAC, promovendo assim uma real ação proativa de segurança.

3.2.1 Características

O conjunto de *patches* que compõem o grsecurity tem como intuito a proteção do espaço de endereçamento (/dev/kmem, /dev/mem e /dev/port), implementar opções para o configurações do modelo RBAC com administração das políticas de acesso através do pacote chamado *gradm*, proteção do *filesystem* (sistemas de arquivos) tal como o pseudo sistema de arquivos /proc, auditoria do Kernel, desabilita também o *framework* LSM por ser considerado pelos desenvolvedores inseguro, pois algum código malicioso pode ser inserido nos módulos de segurança, proteção contra executáveis e rede, suporte para *sysctl* e opções de *log*.

A característica mais importante que já vem por padrão é o *patch* do PaX (*PAge eXecute*) que traz proteção para *flags* de dados na memória como não executável e não gravável. O objetivo principal disso tudo é evitar execução arbitrária de código e injeção de código malicioso, tentando impedir exploração de vulnerabilidade, como *buffer overflow*. Além de fornecer ASLR (*Address Space Layout Randomization*), "mecanismo de segurança que introduz aleatoriedade no processo de alocação dos segmentos de um processo em memória. Esse processo é realizado toda vez que um aplicativo é executado e carregado em memória pelo sistema operacional" (ÁLVARES, 2011). Outros mecânismos complementares ao PaX são utilizados na segurança do sistema, para evitar *stack smashing*:

- PIE (*Position Independent Executables*), execução independente da posição da memória. Este difere do código tradicional na medida em que os acessos às funções são feitos, através de uma tabela de acesso indireto. Auxilia o ASLR a combater *buffer overflow*, e;
- SSP (*Stack Smashing Protector*) é utilizado pelo PaX para detecção de *buffer overflow*, GCC insere código de inicialização nas funções que criam *buffer* na memória.

Conforme SILVA (2008), durante a "execução do processo, quando um *buffer* é criado, o SSP adiciona um valor aleatório secreto, designado por *canary* ao fim do *buffer*". A detecção do *buffer overflow* se dá pela análise deste *canary* que se modificado caracteriza um ataque e o processo seria terminado.

Os níveis de segurança implementados pelo grsecurity são Baixo (*Low*), Médio (*Medium*), Alto (*High*) ou Padrão (*Custom*), neste último onde você pode configurar todas as opções de segurança fornecida pelo grsecurity, nas outras opções você encontra um conjunto específico de seguranças já configuradas.

Como este trabalho visa apenas à implementação do grsecurity, será usado o nível mais alto de segurança, e não será mostrado nenhuma configuração específica, pois cabe ao administrador, saber o que é o mais indicado para o ambiente que ele trabalha, sendo assim indica-se o uso da opção *Custom* e configurando-o de forma específica para seu ambiente.

3.2.2 Preparação do Ambiente

A distribuição utilizada para a instalação do grsecurity será a Arch Linux, que possui o *pacman* (*PACKage MANager*) como gerenciador de pacotes. Para a configuração do grsecurity (PaX) faz necessário o uso das *flags* de compilação:

```
CFLAGS="-march=x86-64 -mtune=generic -O2 -pipe -fPIE -fstack-protector-all -fno-strick-overflow -D_FORTIFY_SOURCE=2"
CXXFLAGS="-march=x86-64 -mtune=generic -O2 -pipe -fPIE -fstack-protector-all -fno-strick-overflow -D_FORTIFY_SOURCE=2"
LDFLAGS="-Wl,-O1,--sort-common,--as-needed,-z,now,-z,relro,--hash-style=gnu"
```

Assim modificando o arquivo /etc/makepkg.conf da distribuição para que todos os pacotes contenham as proteções mencionadas e mais detalhes sobre essas e outras opções de compilação podem ser encontrados no manual do GCC (GNU *project* C *and* C++ *compiler*). Dando continuidade vamos agora extrair alguns dados de antes e depois de toda a compilação da distribuição com as modificações propostas e expondo na continuidade do trabalho.

A compilação do novo kernel já com as *flags* de segurança se dá realizando os seguintes passos:

```
$ tar -zxvf linux-3.0.4.tar.bz2
$ cd linux-3.0.4
$ patch -p1 -i ../grsecurity-2.2.2-3.0.4-201109190917.patch
$ make menuconfig
```

```
# make dep
# make clean
# make bzImage modules modules_install install
```

As opções de segurança podem ser feitas em Security, depois GRSecurity e PaX, conforme o ambiente onde está sendo implementado. Finalizando a instalação do grsecurity com *gradm* e extraindo os dados com o *paxtest* e *paxctl*.

3.3 RESULTADOS

Para o experimento, foram selecionados dois *exploits* (códigos disponibilizados para explorar vulnerabilidade) executáveis para testes, oriundos do site *Exploits DB*[7], sendo que um que mostra se o sistema estava vulnerável para obtenção do *shell* e outro para uma contagem de *overflows* existentes no sistema. Além disso, usamos o comando *paxtest* com o parâmetro *blackhat*, que realizada inúmeros testes para analisar o nível de segurança do sistema operacional.

Segue abaixo os resultados, na primeira coluna os resultados dos sistemas, vulnerável e na segunda do sistema com grsecurity:

Sistema Vulnerável	Sistema com grsecurity
egg mark (designando que o sistema libera a shell após a execução completa do *exploit*).	Não teve permissão para execução, mesmo como *root*.
Matrix multiply sum: s=27665734022509.652344 Total overflows: 198	Não teve permissão para execução, mesmo como *root*.

Tabela 2 resultados da execução dos *exploits*

Os resultados do comando *paxtest*, encontram-se no anexo A.

CONCLUSÃO

Neste trabalho foi visto os controles de acesso largamente utilizados em sistemas operacionais como também em aplicações comerciais e militares, sempre com o intuito de mostrar como melhorar a administração de segurança da informação dentro de qualquer ambiente.

Foi abordado em um nível maior de profundidade o modelo RBAC, por mostrar-se um modelo mais próximo do contexto vivido dentro de uma organização, podendo ser replicado

[7] http://www.exploit-db.com site que amarzena uma quantidade considerável de *exploits* para as mais diversas plataformas e aplicações.

em um sistema operacional com o intuito de implementar políticas de segurança mais eficientes.

Em nosso estudo de caso, utilizou-se o grsecurity em um ambiente *Arch Linux,* para demonstrar que é possível conter ataques ainda desconhecidos pela comunidade de usuários ou fabricante do SO, lançando assim um complemento eficaz para a aplicação de *patches* para falhas de segurança que surgem constantemente. Desta forma, procuramos mostrar que o raciocínio pró-ativo e a implementação de políticas que isolem o potencial de ataques desconhecidos é uma alternativa viável e pode ser de interesse de muitos Administradores de Sistemas.

Como trabalhos futuros, é cogitada a elaboração de um projeto que incorpore o grsecurity e PaX, na distribuição Arch Linux, visto que ainda há uma complexidade considerável para montar um ambiente usando tais ferramentas.

REFERÊNCIAS

ÁLVARES, Marcos. **ALSR - Address Space Layout Randomization.** Disponível em <http://www.scribd.com/doc/46363944/ASLR-Address-Space-Layout-Randomization>. Acesso em 12 de setembro de 2011.

AppArmor. Disponível em < https://apparmor.wiki.kernel.org/index.php/Main_Page>. Acesso em 26 de agosto de 2011.

CERT.br. **Incidentes Reportados ao CERT.br -- Abril a Junho de 2011.** Disponível em <http://www.cert.br/stats/incidentes/2011-apr-jun/total.html>. Acesso em 15 de agosto de 2011.

DRANGER, Stephen; SLOAN, Robert H.; SOLWORTH, Jon A.. **The Complexity of Discretionary Access Control.** Department of Computer Science, University of Illinois, Chicago, 2006.

FERRAIOLO, David F.; KUHN, D. Richard. **Role-Based Access Control.** 15th National Computer Security Conference, Baltimore MD, 1992.

___ IN. **Proposed NIST Standard for Role-Based Access Control.** National Institute of Standards and Technology, Agosto 2001.

___ IN. **Role-Based Access Control (RBAC): Features and Motivations.** Disponível em <http://hissa.nist.gov/rbac/newpaper/rbac.html>. Acesso em 29 de agosto de 2011.

FOX, Michael; GIORDANO, John; STOTLER, Lori; THOMAS Arun. **SELinux and grsecurity: A Case Study Comparing Linux Security Kernel Enchancements.** University of Virginia, Department of Computer Science, 2004.

___IN. **SELinux and grsecurity: A Side-by-Side Comparison of Mandatory Access Control and Access Control List Implementions.**

GRSecurity. Disponível em: <http://grsecurity.net>. Acesso em 20 de setembro de 2011.

Hardened Gentoo Team. **Gentoo Linux Projects - Hardened Gentoo**. Disponível em: <http://www.gentoo.org/proj/en/hardened>. Acesso em 03 de setembro de 2011.

ISO International Organisation for Standards. **Information technology - Security techniques - Evaluation criteria for IT security - Part 1: Introduction and general model**. Suíça, 2009.

KRAHMER, Sebastian. **Hardened *OS explotation techniques**. Disponível em <http://eldorado.tu-dortmund.de:8080/bitstream/2003/22817/1/DIMVA2004-SP-Krahmer.pdf>. Acesso em 6 de Agosto de 2011.

LENTO, Otávio Botelho; FRAGA, Joni da Silva; LUNG, Lau Cheuk. **A Nova Geração de Modelos de Controle de Acesso em Sistemas Computacionais.** Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 6, 2006.

MACHADO JUNIOR, Dorival Moreira. **Proposta de *Hardening* em Conformidade com a ISO 27001 para um Firewall em Linux com Balanceamento de Carga**. Universidade Federal de Lavras, Minas Gerais, 2008.

MATEUS, Hudson Flávio Vieira. **Buffer Overflow: Teoria e Exploração.** Universidade Federal de Lavras. Minas Gerais, 2011.

MATTOS, Cristiano Lincoln de Almeida. **Sentinel: um engenho Java para controle de acesso RBAC.** Universidade Federal de Pernambuco, Recife, Agosto de 2003. Disponível em <http://www.scribd.com/doc/51042725/controle-de-acesso>. Acesso em 13 de Agosto de 2011.

MOFFET, Jonathan D.. **Specification of Management Policies and Discretionay Access Control.** Department of Computing, Imperial College of Science Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, UK.

NIST/ITL National Institute of Standards and Technology/InformationTechnology Laboratory. **An Introdution to Roled-Based Access Control.** Bulletin, Dezembro, 1995. Disponível em <http://csrc.nist.gov/groups/SNS/rbac/documents/design_implementation/Intro_role_based_access.htm>. Acesso em 4 de setembro de 2011.

NSA National Security Agency of USA. **Department of Defense Standard. Orange Book Trusted Computer System Evaluation Criteria.** Dezembro 1995.

__IN. **Role Based Access Control (RBAC) e Role Based Security**. Disponível em <http://csrc.nist.gov/groups/SNS/rbac/>. Acesso em 25 de agosto de 2011.

OSBORN, Sylvia; SANDGU, Ravi; MUNAWER, Qamar. **Configuring Rol-Based Access Control To Enforce Mandatory and Discretionary Access Control Policies.** *Proceedings,*

5th ACM Workshop on Role Based Access Control, July 26-27, 2000, Berlin, pp.47-63 - first public draft of the NIST RBAC model and proposal for an RBAC standard.

PASSOS, Clayton Kendy Nakahara. **Estratégias para Firewall.** Disponível em < http://clayton.bs2.com.br/artigos/EstrategiasParaFirewall.pdf>. Acesso em 5 de setembro de 2011.

RAYMOND, Erick Steven. **The Art of Unix Programming.** 2003. Disponível em <http://softwarelibre.unsa.edu.ar/programacion/taoup.pdf>. Acesso em 21 de setembro de 2011.

SANDHU, Ravi; FERRAIOLO, David; KUHN, Richard. **The NIST Model for Role-Based Access Control: Towards A Unified Standard.** Laboratory for Information Security Technology (LIST) George Mason University and Information Technology Laboratory, National Institute of Standards and Technology (NIST), 2000.

SELinux. Disponível em <http://www.nsa.gov/research/selinux/>. Acesso em 21 de setembro de 2011.

SILVA, Gleydson Mazioli da. **Guia Foca Linux Intermediário.** Acesso em 25 de agosto, 2011. Disponível em <http://www.guiafoca.org>.

__IN. **Guia Foca Linux Avançado.** Acesso em 26 de agosto, 2011. Disponível em <http://www.guiafoca.org>.

SILVA, Vitor Manuel Brandão Moreira da. **Análise e Concepção de Servidores Linux Seguros**. Faculdade de Engenharia da Universidade do Porto, Portugal, 2008.

Smack. Disponível em < http://www.mjmwired.net/kernel/Documentation/Smack.txt>. Acesso em 21 de setembro de 2011.

Tomoyo. Disponível em < http://tomoyo.sourceforge.jp/>. Acesso em 21 de setembro de 2011.

WRIGHT, Chris; COWAN, Crispin; MORRIS, James; SMALEY, Stephen; KROAH-HARTMAN, Greg. **Linux Security Modules: General Security Support for the Linux Kernel**. Proceedings of the 11th USENIX Security Simposium, Baltimore, São Francisco, Califórnia, USA 5-9 Agosto 2002. Disponível em: <http://www.usenix.org/event/sec02/full_papers/wright/wright.pdf>. Acesso em 07 de setembro de 2011.

YAMAZAKI, Tiago Jun. **Estudo sobre Vulnerabilidade de Strings de Formação.** Universidade Federal do Rio Grande do Sul, Instituto de Informática. Porto Alegre, Novembro de 2009.

ZUCCO, Jerônimo Cleberson**.** ***Hardening*** **Linux Usando Controle de Acesso Mandatório**. Universidade de Caxias do Sul, Caxias do Sul, 2008.

ANEXO A

Segue os resultados do comando *paxtest* o primeiro resultado é da instalação padrão da distrbuição e o segundo, é da distribuição toda recompilada com as *flags* de segurança utilizadas por padrão com o grsecurity.

Primeiro o resultado do comando *uname -a*, mostrando onde foi realizado os testes:

uname –a

Linux archgrsec 3.0-grsec-ARCH #1 SMP PREEMPT Tue Sep 20 14:30:23 UTC 2011 x86_64 Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz GenuineIntel GNU/Linux

PaXtest - Copyright(c) 2003,2004 by Peter Busser <peter@adamantix.org>

Released under the GNU Public Licence version 2 or later

Mode: blackhat

Linux archgrsec 3.0-grsec-ARCH #1 SMP PREEMPT Tue Aug 30 07:33:25 UTC 2011 x86_64 Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz GenuineIntel GNU/Linux

Executable anonymous mapping	: Vulnerable
Executable bss	: Vulnerable
Executable data	: Vulnerable
Executable heap	: Vulnerable
Executable stack	: Killed
Executable shared library bss	: Vulnerable
Executable shared library data	: Vulnerable
Executable anonymous mapping (mprotect)	: Vulnerable
Executable bss (mprotect)	: Vulnerable
Executable data (mprotect)	: Vulnerable
Executable heap (mprotect)	: Vulnerable
Executable stack (mprotect)	: Vulnerable
Executable shared library bss (mprotect)	: Vulnerable
Executable shared library data (mprotect)	: Vulnerable
Writable text segments	: Vulnerable
Anonymous mapping randomisation test	: 12 bits (guessed)
Heap randomisation test (ET_EXEC)	: 13 bits (guessed)
Heap randomisation test (PIE)	: 16 bits (guessed)
Main executable randomisation (ET_EXEC)	: No randomisation
Main executable randomisation (PIE)	: 12 bits (guessed)
Shared library randomisation test	: 10 bits (guessed)
Stack randomisation test (SEGMEXEC)	: 19 bits (guessed)
Stack randomisation test (PAGEEXEC)	: 19 bits (guessed)
Return to function (strcpy)	: Vulnerable
Return to function (memcpy)	: Killed

Return to function (strcpy, PIE) : Vulnerable
Return to function (memcpy, PIE) : Killed

PaXtest - Copyright(c) 2003,2004 by Peter Busser <peter@adamantix.org>

Released under the GNU Public Licence version 2 or later

Mode: blackhat

Linux archgrsec 3.0-grsec-ARCH #1 SMP PREEMPT Tue Sep 20 14:30:23 UTC 2011 x86_64 Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz GenuineIntel GNU/Linux

Executable anonymous mapping : Killed
Executable bss : Killed
Executable data : Killed
Executable heap : Killed
Executable stack : Killed
Executable shared library bss : Killed
Executable shared library data : Killed
Executable anonymous mapping (mprotect) : Killed
Executable bss (mprotect) : Killed
Executable data (mprotect) : Killed
Executable heap (mprotect) : Killed
Executable stack (mprotect) : Killed
Executable shared library bss (mprotect) : Killed
Executable shared library data (mprotect) : Killed
Writable text segments : Killed
Anonymous mapping randomisation test : 33 bits (guessed)
Heap randomisation test (ET_EXEC) : 13 bits (guessed)
Heap randomisation test (PIE) : 40 bits (guessed)
Main executable randomisation (ET_EXEC) : No randomisation
Main executable randomisation (PIE) : 32 bits (guessed)
Shared library randomisation test : 33 bits (guessed)
Stack randomisation test (SEGMEXEC) : 40 bits (guessed)
Stack randomisation test (PAGEEXEC) : 40 bits (guessed)
Return to function (strcpy): paxtest: return address contains a NULL byte.
Return to function (memcpy) : Killed
Return to function (strcpy, PIE) : paxtest: return address contains a NULL byte.
Return to function (memcpy, PIE) : Killed