Islan Nascimento Rocha

Análise e utilização da ferramenta Ksplice

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Joaquim Uchôa

Lavras
Minas Gerais - Brasil
2011

Islan Nascimento Rocha

Análise e utilização da ferramenta Ksplice

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *30 de Abril de 2011*

Prof. Arlindo Follador Neto

Prof. Tales Heimfarth

Prof. Joaquim Uchôa
(Orientador)

Lavras
Minas Gerais - Brasil
2011

Dedico a todos que estiveram envolvidos durante a produção deste trabalho.

Agradecimentos

Agradeço aos professores, que através de suas exigências, nos fazem alunos melhores.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A computação vive uma nova era, voltando a concentrar capacidade de processamento e armazenamento de informações em *hardware* especializado. Aliado a esta nova era, está o tema T.I. verde que tem contribuído para a adoção cada vez maior da virtualização e visa a otimização na utilização de recursos energéticos entre outros. Como consequência, inúmeras máquinas virtuais tem sido consolidadas em um único ambiente, aumentando a complexidade para se manter a segurança e disponbilidade. A inovação proposta pelo *software* Ksplice, consiste em manter o ambiente computacional livre de vulnerabilidades de *kernel*, sem que para isso seja necessário comprometer as exigências por disponibilidade, cada vez mais presente nestes ambientes, pois ele permite a atualização do *kernel* sem a necessidade de reinicialização de todo sistema operacional. Este trabalho fará uma análise da ferramenta Ksplice e demonstrará sua utilização em um ambiente vunerável.

Palavras-Chave: Ksplice; Vulnerabilidades; Virtualização; Alta Disponibilidade.

Capítulo 1

Introdução

1.1 Motivação

Atualmente, a crescente adoção da virtualização em ambientes coorporativos e *datacenters* tem ampliado o poder de dano de ataques que exploram vulnerabilidades do *kernel*. Além disso, o número de vulnerabilidades de *kernel* é expressivo, o que faz com que os administradores destes ambientes se preocupem com a segurança, pois a exploração dessas falhas podem não só ter como objetivo a obtenção de informações mas também a indisponibilização de serviços através de ataques de DoS[1] ou ainda *defacements*[2], no caso de servidores *web*.

Aliado às vulnerabilidades, há o surgimento de novas tecnologias, como por exemplo, o recurso *Kernel SamePage Merging (KSM)* (RED HAT INC., 2010), que permite que a solução de virtualização da Red Hat, KVM, faça a desduplicação de porções de memória com conteúdo idêntico. A partir da utilização desta tecnologia, torna-se possível a execução, em um *hardware* com 16 GB de memória RAM, de 52 máquinas virtuais com Windows XP, cada uma com 1GB de memória RAM disponível (HESS, 2010).

A utilização deste tipo de tecnologia demonstra claramente a amplificação dos danos que podem ser causados por um ataque à uma máquina hospedeira vunerável, pois, levando-se em conta o anúncio da Red Hat, correria-se o risco de que um ataque indisponibilizasse 52 máquinas de uma só vez.

[1] *Denial of Service*, ou Negação de serviço - trata-se do ataque cujo objetivo é a derrubada de um serviço, tornando-o inacessível.

[2] Ataque cujo objetivo é a descaracterização de sites *web*

Além da problemática relacionada à segurança, há também as questões ligadas aos níveis de serviços (SLAs) acordados durante a disponibilização de um serviço/servidor. Espera-se com este trabalho demonstrar como a ferramenta Ksplice pode colaborar para que um ambiente computacional possa ser mantido sempre atualizado sem comprometer, por exemplo, os tempos previstos para manutenção em contratos de alta disponibilidade.

1.2 Objetivos

O foco deste trabalho é apresentar a ferramenta Ksplice, cujo método inovador vem incrementar a segurança em ambientes computacionais baseados em Linux. Pretende-se detalhar e demonstrar a utilização deste *software*, bem como elencar as vantagens de sua adoção e também apresentar exemplos de ambientes que podem ser beneficiados por ele.

1.3 Metodologia

O entendimento do funcionamento da ferramenta será conseguido através do detalhamento de seu *design*, lançando-se mão de explicações suscintas.

Para demonstrar a eficácia da ferramenta, será utilizado um ambiente com diferentes vulnerabilidades de *kernel*, das quais uma será explorada antes da utilização da ferramenta e também posteriormente a aplicação da atualização.

1.4 Estrutura dos tópicos

No Capítulo 2 serão abordados todos os conceitos necessários para o entendimento do Ksplice, seu histórico e seu *design*. Já no Capítulo **??** serão apresentados detalhes do ambiente utilizado nos testes, entre eles, detalhes do *hardware*, da vulnerabilidade e do *exploit* utilizado. Os testes e resultados estarão no Capítulo 4 e a conclusão do trabalho no Capítulo 5.

Capítulo 2

Fundamentação Teórica

2.1 Principais Tipos de Linguagens de programação

O entendimento dos tipos de linguagens de programação são fundamentais para a compreensão da diferença existente entre os tipos de códigos gerados por estas linguagens, e consequentemente, da camada na qual a ferramenta abordada neste trabalho atua. Vale lembrar que o *kernel* do Linux utiliza, em sua maior parte, uma linguagem de programação compilada, C, portanto, como veremos adiante, o *kernel* assim como seus módulos, após compilados, são códigos objeto, comumente chamados de binários.

Em ambos os tipos de linguagens, o código gerado pelo programador é o código fonte, ou *source code*, que é facilmente compreendido se comparado ao código resultante da compilação ou interpretação, código objeto, que será efetivamente entendido e executado pelo *hardware*.

Nas linguagens compiladas o código fonte é enviado para o compilador, que gera um código de montagem, geralmente em Assembly, e este código por sua vez é transformado em código objeto, que será efetivamente compreendido pelo processador responsável pela execução das instruções (AHO; SETHI; ULLMAN, 1995).

Em liguagens interpretadas há a geração de um código intermediário, conhecido por *bytecode*, este *bytecode* é interpretado por uma aplicação que é análoga a um processador, que geralmente é chamada de máquina virtual.

Entendida a diferença entre os tipos de linguagem, pode-se focar nas linguagens compiladas, pois como já mencionado, o *kernel* do Linux é escrito em sua maior parte na linguagem C, uma linguagem compilada.

Ainda sim faz-se necessário o entendimento da diferença entre código fonte e código objeto, pois apesar de terem nomes sugestivos, podem causar certa confusão. Entende-se por código fonte a instrução passada ao compilador, geralmente em uma linguagem de alto nível, que possui um nível de abstração mais elevado e mais próximo à linguagem humana, como é o caso da linguagem C. Enquanto por código objeto, entende-se ser o produto da compilação, que por apresentar-se em uma linguagem de baixo nível e dificultar exponencialmente seu entendimento por um humano é considerada uma linguagem de baixo nível ou linguagem de máquina.

2.2 *Kernel* do Linux

Kernel Linux é o núcleo do sistema operacional GNU/Linux cujas funções fundamentais incluem tratamento de interrupções, escalonamento de processos e gerenciamento de memória (PEREIRA, 2006).

Seu desenvolvimento se iniciou em abril de 1991 por Linus Torvalds, com o intúito inicial de ser apenas um *hobbie*. Em setembro deste mesmo ano foi lançada a sua primeira versão e, a partir de então, passou a receber colaboração de milhares de desenvolvedores (TIBET, 2001).

Há três tipos principais de *kernel*: monolítico, *microkernel* e híbrido. O *kernel* Linux é monolítico pois é implementado como um grande processo executando em um único espaço de endereçamento. Porém, apresenta características de *microkernel* como, por exemplo, o suporte a *threads*, capacidade de carregar dinamicamente bibliotecas separadas do *kernel* (*kernel modules*) e por possuir um projeto modular com *kernel* preemptivo.

Suas versões dividem-se entre estáveis e em desenvolvimento, e podem ser diferenciadas através do número de versão, composto por três números separados por ponto. O primeiro algarismo é a *release* maior, o segundo é a *release* menor e o último é a revisão (*patchlevel*). A diferenciação entre versão estável e desenvolvimento é feita através do algarismo da *release* menor: quando ímpar, trata-se de um *kernel* em desenvolvimento (LOVE, 2010).

2.3 *Patch de Kernel*

Patches de kernel, como a própria tradução do termo *patch* sugere, trata-se de um remendo no código fonte original.

Sua motivação pode ser para a correção de um *bug*, vulnerabilidade ou ainda, alterações de estruturas que possibilitem melhorias de desempenho.

Os *patches* atuam no que se pode classificar como sendo camada do código fonte, pois alteram exatamente o código fonte original do *kernel*. Até o surgimento da ferramenta Ksplice, a aplicação de *patches* podia ocorrer de duas maneiras:

- Aplicando-se o *patch* ao código fonte original, compilando-o e carregando-o.
- Baixando-se do site do mantenedor da distribuição GNU/Linux a ser atualizada um binário já compilado para a arquitetura da máquina na qual a distribuição é utilizada.

Ambos métodos geram atividades que mesmo sendo executadas por especialistas, podem resultar em horas de trabalho sobre o ambiente e consequentemente horas de indisponibilidade.

Diferentemente dos métodos tradicionais, a ferramenta Ksplice atua na camada de código objeto. Isso permite a economia do tempo que seria gasto com recompilação de código ou *download* do binário já compilado, e também a economia do tempo gasto na reinicialização requerida por estes métodos (ARNOLD; KAASHOEK, 2009).

2.4 Vulnerabilidades

Entende-se por vulnerabilidade falhas de código que podem acarretar algum tipo de comprometimento em um ambiente computacional (NAKAMURA; GEUS, 2003). No caso específico do sistema operacional Linux, pode-se elencar alguns dos principais sintomas causados pela exploração destas vulnerabilidades:

- Acesso não autorizado ao sistema;
- Negação de serviço;

- Aumento de privilégios.

Atualmente, há diversas entidades que mantém listas de vulnerabilidades conhecidas, cuja função, além da divulgação, é fornecer subsídios para que administradores saibam como tratar seus ambientes, bem como permitir aos mantenedores do código vulnerável tratar a falha divulgada.

Entre os sites com esta função está o conhecido CVE *(Common Vulnerabilities and Exposures)*[1], que provê um dicionário com as vulnerabilidades mais conhecidas de uma forma padronizada, acrescentando informações que auxiliam na obtenção de mais conhecimento sobre a vulnerabilidade em questão.

Já em sites como o *Security Focus*[2], é possível encontrar, além de informações sobre a vulnerabilidade, soluções para sanar o problema detectado.

Baseado em informações de outro *site* especializado em segurança, Secunia[3], pode-se ter uma melhor percepção do volume de vulnerabilidades catalogadas desde o lançamento da release 2.6 do kernel, em 17 de dezembro de 2003.

Até inicio de 2011, foram catalogadas pela Secunia, 260 vulnerabilidades relacionadas à versão 2.6.x do *kernel* do Linux. Destas, 42 permitiam exploração remota, 190 exploração local no sistema e 28 permitiam exploração remota em uma rede local (SECUNIA, 2011).

Entre os principais sintomas causados pela exploração destas vunerabilidades estão a negação de serviço, que era possível através da exploração de 112 das 260 vulnerabilidades, e a obtenção de privilégios, que era possível em 47 das falhas.

2.5 Acordo de Nível de Serviço (ANS)

Mais conhecido por sua sigla, do inglês *Service Level Agreement*, o SLA é a materialização das expectativas dos usuários quanto a disponibilidade e suporte aos serviços fornecidos. Também está relacionada às necessidades que o fornecedor terá, geralmente convertida em valores, para prover o serviço de acordo com a expectativa apresentada.

[1] http://cve.mitre.org/index.html
[2] http://www.securityfocus.com/bid
[3] http://secunia.com

Tabela 2.1: Classes de Disponibilidade e *Downtime* por ano

Disponibilidade %	*Downtime por ano*
99	3,65 Dias
99.9	8,76 Horas
99.99	52,6 Minutos
99.999	5,26 Minutos
99.9999	30 Segundos

A definição do SLA deve levar em consideração indicadores que sejam específicos, mensuráveis, atingíveis, além de realistas e orientados pelo tempo (SOUZA, 2007).

Entendido o conceito de SLA, torna-se fácil a compreensão da relação entre o Acordo de Nível de Serviço e ambientes de alta disponibilidade. Toda a infraestrutura tecnológica em torno do provimento de um serviço está sujeita ao nível de serviço acordado em contrato entre cliente e fornecedor. Em ambientes de alta disponibilidade, o tempo de parada, ou *downtime*, disponível para manutenções agendadas ou mesmo indisponibilidade por falhas, são extremamente limitados e a extrapolação destes limites compromete o nível de serviço acordado.

O nível de disponibilidade esperado pelo ambiente computacional geralmente é apresentado em porcentagens, mas pode ficar mais claro se for apresentado o tempo máximo de indisponibilidade que este ambiente pode ter em um ano. Este calculo pode ser feito utilizando-se a equação 2.1.

$$DowtimePorAno(Minutos) = (1 - \frac{Disponibilidade}{100}) * 365 * 24 * 60 \qquad (2.1)$$

A partir das porcentagens que classificam ambientes de alta disponibilidade, acima de 99%, é possível confeccionar um quadro que demonstra o tempo disponível para indisponibilidade do ambiente, conforme pode ser visto na Tabela 2.1.

2.6 Ksplice

O Ksplice[4] é uma extensão *open source* do *kernel*, desenlvolvida pela Ksplice Inc. e licenciada sobre GPLv2, que permite ao administrador do sistema aplicar *patches* de segurança no *kernel* em execução sem a necessidade de reinicializações. Atualmente, está disponível para as arquiteturas x86 32 *bits* e x86 64 *bits* (KSPLICE INC., 2011b).

A inovação proposta por esta ferramenta reside na capacidade que ela tem de, a partir dos *patches de kernel*, que atuam a nível de código fonte, ser capaz de criar uma atualização de código a nível de código objeto.

Atualmente, a empresa desenvolvedora oferece a ferramenta *open source* Ksplice e um produto diferenciado chamado Ksplice Uptrack. Este último é uma versão suportada pela empresa, que oferece o serviço de análise e a entrega dos *patch*es customizados, quando isso é necessário.

Nada impede a utilização da versão aberta para a geração e aplicação das atualizações, porém é recomendável a aquisição e utilização do Ksplice Uptrack em ambientes de produção, já que as atualizações são previamente analisadas pela equipe da empresa Ksplice Inc.

2.6.1 Histórico

A idéia por trás da ferramenta Ksplice surgiu no verão americano de 2006, quando o seu criador, Jeffrey Arnold, era membro do *SIPB (Student Information Processing Board)*[5] e também administrador de um conhecido site de *scripts* do *MIT (Massachusetts Institute of Technology)*[6].

Neste período ele necessitou atualizar o servidor do *site* para restringir uma vulnerabilidade do *kernel* descoberta no meio da semana. Mas para evitar transtorno aos usuários esta atualização foi agendada para a madrugada de domingo afetando assim o menor número possível de usuários (KSPLICE INC., 2011a).

Como era de ser esperar, a falha que seria corrigida pela atualização agendada foi explorada no período que antecedeu a atualização. A partir deste episódio, Jeffrey passou a questionar o porquê de uma atualização de *kernel* ser tão pertubadora. Então propôs-se a solucionar o problema em seu trabalho de mestrado.

[4] http://www.ksplice.com/
[5] http://sipb.mit.edu
[6] http://scripts.mit.edu/

Como a necessidade por uma solução como essa sempre existiu, em Junho de 2008, Jeffrey Arnold, Tim Abbott e Anders Kaseorg fundaram a Ksplice Inc.

Atualmente, eles e mais alguns estudantes de computação do MIT trabalham para disponibilizar as novas atualizações trazidas pelos *patch*es recentes.

2.6.2 *Design* do Ksplice

O Ksplice foi desenvolvido para utilizar os *patches* providos pela comunidade que em sua maioria podem ser transformados em *hot updates*[7] sem a necessidade de serem reescritos por algum programador, pois não fazem alterações semânticas em estruturas persistentes do *kernel*.

Nos casos em que o *patch* faz alterações nestas estruturas persistentes, há a necessidade de reescrita de código do *patch* a fim de adequá-lo ao que é esperado pela ferramenta para que então ela seja capaz de fazer consistentemente a atualização.

Tradicionalmente, *patches* de segurança ou de correções de *bugs* para o *kernel* do Linux procuram fazer o mínimo de alterações possível e por isso, são raros os casos em que há modificações que necessitem chegar a ponto de alterar estruturas persistentes, já que estas estruturas costumam ser definidas e implementadas, nos períodos iniciais de projeto (KSPLICE INC., 2011b).

Isto pode ser comprovado pela quantidade de intervenções necessárias nos *patch*es providos entre 2005 e Maio de 2008, em que apenas 8 dos 64 *patches* tiveram que ser reescritos antes de serem utilizados pelo Ksplice. As Tabelas 2.2 e 2.3 trazem a listagem dos *patches* lançados entre 2005 e 2008 e os que tiveram necessidade de serem modificados antes de serem utilizados pela ferramenta.

A ferramenta atua na camada de código objeto e basicamente faz a substituição completa de uma função alterada por um *patch*, ligando o novo código da função dentro do *kernel*. Para isso, é colocado um "*jump*" no *kernel* em execução na memória, no início da função obsoleta, apontando para o novo código da função.

[7] Também conhecido pelo termo *hot patching*, trata-se da capacidade de aplicação de *patches* sem a necessidade de reinicializações do sistema operacional e/ou do *software* que está sendo atualizado.

Tabela 2.2: Vulnerabilidades corrigidas pela ferramenta Ksplice entre 2005 e Maio de 2008

Cdigo da Vulnerabilidade	*Data do Patch*	*PatchReescrito*
CVE-2005-1263	2005-05-11	
CVE-2005-1264	2005-05-13	
CVE-2005-1589	2005-05-14	
CVE-2005-2456	2005-07-26	
CVE-2005-3276	2005-07-27	
CVE-2005-2500	2005-08-10	
CVE-2005-2492	2005-09-19	
CVE-2005-3179	2005-10-03	Sim
CVE-2005-3180	2005-10-04	
CVE-2005-2709	2005-11-04	Sim
CVE-2005-4639	2005-11-08	
CVE-2005-3784	2005-11-10	
CVE-2005-4605	2005-12-30	
CVE-2006-0095	2006-01-06	
CVE-2006-0457	2006-02-03	
CVE-2006-2071	2006-04-12	Sim
CVE-2006-1524	2006-04-17	
CVE-2006-1056	2006-04-20	Sim
CVE-2006-1863	2006-04-21	
CVE-2006-1864	2006-05-15	
CVE-2006-0039	2006-05-19	
CVE-2006-1857	2006-05-19	
CVE-2006-1858	2006-05-19	
CVE-2006-1343	2006-05-28	
CVE-2006-2935	2006-07-10	
CVE-2006-2451	2006-07-12	
CVE-2006-3626	2006-07-14	
CVE-2006-3745	2006-08-22	
CVE-2006-5751	2006-11-20	
CVE-2006-6304	2006-12-06	
CVE-2006-5753	2007-01-05	Sim

Tabela 2.3: Continuação - Vulnerabilidades corrigidas pela ferramenta Ksplice entre 2005 e Maio de 2008

Cdigo da Vulnerabilidade	*Data do Patch*	*PatchReescrito*
CVE-2006-6106	2007-01-08	
CVE-2007-0958	2007-01-26	
CVE-2007-1217	2007-02-28	
CVE-2007-0005	2007-03-06	
CVE-2007-1000	2007-03-09	
CVE-2007-1730	2007-03-16	
CVE-2007-1734	2007-03-28	
CVE-2007-2480	2007-04-30	
CVE-2007-1353	2007-05-05	
CVE-2007-2875	2007-05-09	
CVE-2007-3105	2007-07-19	
CVE-2007-3851	2007-08-07	Sim
CVE-2007-3848	2007-08-17	
CVE-2007-3740	2007-09-13	
CVE-2007-4571	2007-09-17	Sim
CVE-2007-4308	2007-11-07	
CVE-2007-5904	2007-11-13	
CVE-2007-6206	2007-11-28	
CVE-2007-6417	2007-11-28	
CVE-2007-6063	2007-12-01	
CVE-2007-6434	2007-12-04	
CVE-2007-5966	2007-12-07	
CVE-2008-0001	2008-01-12	
CVE-2008-0007	2008-02-02	Sim
CVE-2008-0009	2008-02-08	
CVE-2008-0600	2008-02-10	
CVE-2008-1367	2008-03-05	
CVE-2008-1675	2008-04-20	
CVE-2008-1375	2008-05-01	
CVE-2008-2148	2008-05-01	
CVE-2008-1669	2008-05-06	
CVE-2008-1294	2008-05-08	
CVE-2008-1673	2008-06-04	

Quando aplicados, estes *updates* levam em média 0.7 milesegundos para serem concluídos, com a vantagem de não serem necessárias reinicializações e consequentemente com menores impactos para o negócio (ARNOLD; KAASHOEK, 2009). Além disso, este procedimento demanda pouquíssima memória para o armazenamento do novo código das funções e alguns ciclos de CPU a mais devido aos *jumps* que foram inseridos nas funções originais apontando para as substitutas.

Visando facilitar a resolução de alguns desafios comuns em técnicas de *hot updates*, a ferramenta foi desenhada para trabalhar na camada de código objeto. Entre os desafios estão: geração do código substituto, resolução de símbolos no código substituto e verificação do estado pós atualização.

2.6.2.1 Geração do Código Substituto

Para gerar o código substituto o sistema de *hot update* deve identificar o que foi alterado após a aplicação do *patch*. O Ksplice resolve esta questão utilizando a técnica de diferenciação *pre-post*, comparando o binário do *kernel* antes da aplicação do *patch* (pre) e o binário resultante depois da sua aplicação (*post*).

A técnica *pre-post* adotada pelo Ksplice atua na camada de código objeto, pois caso atuasse na camada de códico fonte necessitaria descobrir e alterar todos os pontos do código onde ocorrem as chamadas às funções alteradas pelo *patch*.

Para entender qual a alteração feita pelo *patch*, o Ksplice gera duas compilações do *kernel*, uma a partir do código fonte original do *kernel* que será atualizado e outra a partir do código fonte atualizado pelo *patch*. Além disso, para que a comparação se torne prática, a ferramenta faz uso de algumas opções de compilação que evitam que o código objeto contenham informações sobre onde as funções e suas estruturas de dados estão posicionadas na memória (ARNOLD; KAASHOEK, 2009).

As opções de compilação são: `-ffunction-sections` e `-fdata-sections` e são encontradas no compilador GCC. A primeira opção instrui ao GCC para colocar cada função em uma seção específica com o próprio nome da função no arquivo que será tratado pelo *linker*, ao invés de incluí-la em uma grande seção com todas as funções do código compilado. A segunda opção é análoga à primeira, porém, ao invés das funções o que será organizado em seções serão os dados (funções globais e estáticas).

A utilização destas opções garante que funções que não foram alteradas pelo *patch* conterão o mesmo código objeto nas compilações *pre* e *post*, o que facilitará a extração das funções que realmente foram modificadas.

A partir da comparação entre as compilações será gerado um módulo de *kernel* primário. Este módulo terá acrescentada algumas informações pelo processo que veremos na Subseção 2.6.2.2.

2.6.2.2 Resolução de Símbolos no Código Substituto

O *kernel* do Linux mantém uma estrutura chamada de tabela de símbolos do *kernel (Kernel Symbol Table)* que contém a referência de todos os recursos carregados em memória durante sua execução, como por exemplo funções e variáveis. Esta estrutura, durante o funcionamento do *kernel* é encontrada em /proc/kallsyms e basicamente sua estrutura é composta pelo nome do recurso, função ou variável, e o endereço de memória que contém esta estrutura (CORBET; KROAH-HARTMAN; RUBINI, 2005).

Como foi visto na Subseção 2.6.2.1, há a compilação de dois *kernels*, um com o código fonte original e outro com o código alterado pelo *patch*, porém o módulo primário gerado a partir da comparação das compilações do *kernel* não possui a referência de memória em sua estrutura pois ainda não foi "carregado". Para solucionar este problema, é feita uma comparação entre o *kernel* em execução e o objeto gerado pela compilação antes da aplicação do *patch*. Esta comparação verifica a tabela de simbolos do *kernel* em execução e adiciona as referências de memória ao código objeto gerado pela compilação do código fonte original.

2.6.2.3 Verificação de Segurança das Comparações de Código

Para que o processo de geração do módulo primário seja confiável, é preciso garantir que os *kernels* compilados, utilizados na comparação *pre* e *post*, sejam compatíveis com a versão do *kernel* em execução. O Ksplice possui uma etapa que faz exatamente esta verificação entre as versões dos *kernels* utilizados nas compilações.

Capítulo 3

Metodologia

Neste Capítulo, serão descritas algumas características do ambiente que foi utilizado nos testes, entre eles, o *hardware*, o Sitema Operacional e outros aplicativos que foram necessários. Também serão abordados detalhes da instalação do aplicativo na máquina virtual utilizada nos testes.

Pretende-se com a preparação deste ambiente, possibilitar os testes no próximo capítulo, que consistirão na exploração de uma vulnerabilidade específica antes e após a atualização do *kernel* utilizando o Ksplice.

3.1 Ambiente

3.1.1 *Hardware*

Utilizou-se nos testes um computador com processador Turion(tm) 64 X2 *Mobile Technology* TL-58 (1900MHZ) com 1MB de cache L2, 2GB de memória RAM DDR2 e *hard disk* SATA 5.400 rpm de 160GB.

3.1.2 Sistema Operacional

Os testes foram realizados em uma máquina virtual contendo a distribuição Ubuntu Lucid (10.04) recém instalada sem a atualização dos pacotes de segurança mais recentes. Este é um cenário bastante comum, já que, no momento da redação deste trabalho, utilizou-se a versão mais atual desta distribuição e na maioria das vezes

durante a instalação inicial do Sistema Operacional não são baixados e instalados os pacotes de segurança já disponíveis.

3.1.3 Conexão com *Internet*

A ferramenta utilizada neste trabalho, por ser a versão *open source* suportada pela empresa desenvolvedora, já disponibiliza os pacotes com os módulos que contém as diferenças entre o *kernel* em execução e o *kernel* com os *patches* aplicados. Isso torna a atualização menos demorada, porém, ela dependerá de uma conexão com a internet.

Não há a necessidade de conexão de alta velocidade, pois em média, os pacotes de atualizações tem aproximadamente 100KB cada. Sendo assim, a conexão com a Internet não deve ser considerada um gargalo, pois em um ambiente virtualizado ou mesmo coorporativo há uma certa homogeneidade nas versões do Sistema Operacional instalado e isso pode facilitar com que tais atualizações possam ser intermediadas por um *proxy*. Isso evita que o pacote de atualização seja baixado tantas vezes quantas forem as estações a serem atualizadas.

Nestes testes a conexão utilizada foi um link ADSL de 600kbps.

3.1.4 Outros Aplicativos

Na máquina virtual, foram instalados o Ksplice Uptrack, que permitirá a aplicação do *path de kernel* que suprime a vulnerabilidade que será explorada, além de suas dependências (`curl, python-yaml e libyaml`) e as ferramentas necessárias para a compilação do *exploit* para a vulnerabilidade CVE-2959 que foi utilizado nos testes, como por exemplo o compilador GCC.

3.2 Instalação

Conforme foi dito na Seção 2.6, a Ksplice Inc. disponibiliza a ferramenta em dois produtos distintos: Ksplice e Ksplice Uptrack. O primeiro, é a versão de código aberto, licenciada sob GPLv2, que permite a construção do *hot update* a partir do *patch do kernel* disponível, por exemplo, no site oficial do *kernel*[1]. O segundo produto trata-se da opção paga, que disponibiliza os *patches* próprios após

[1] http://kernel.org/

uma análise dos *patches* originais do site do *kernel* pelos engenheiros da Ksplice Inc. Em ambiente de alta criticidade, que dependem de estabilidade e seguraça, é recomendável a utilização da ferramenta paga, já que pode haver *patches* que não podem ser transformados automaticamente em *hot updates*.

O primeiro passo deve ser o *download* da ferramenta no site da Ksplice Inc de acordo com o Sistema Operacional, neste caso foi baixado o pacote .deb para o Ubuntu Lucid 10.04. A versão disponibilizada pela Ksplice durante a produção deste trabalho estava em 1.1.3.

O segundo passo será a instalação dos pacotes `curl, python-yaml e libyaml`, dependências necessárias para a instalação do Ksplice. Como foi utilizado o Ubuntu, pode-se utilizar o gerenciador de pacotes apt-get para a instalação destas dependências.

O próximo e último passo é a instalação do pacote baixado no primeiro passo. O que pode ser feito através do comando `dpkg -i <nome do pacote>`.

Após a instalação, é criado o arquivo de configuração /etc/uptrack/uptrack.conf. Entre as opções deste arquivo estão as mostradas na Tabela **??**.

Tabela 3.1: Opções do arquivo de configuração `/etc/uptrack/uptrack.conf`

Opção	**Descrição**
`https_proxy`	Deve ser definida quando se deseja utilizar um *proxy*. Quando definido como `None` o Ksplice não utilizará o *proxy*, do contrário o formato de preenchimento desta opção deverá ser `[protocol://][username:password@]<host>[:port]` .
`gconf_proxy_lookup`	Quando setada pra `yes|Yes` utiliza a configuração de *proxy* do Sistema Operacional. Esta configuração pode ser definida através do `gconf-editor`.
`install_on_reboot`	Quando habilitada (`yes|Yes`) reaplica as atualizações de *kernel* automaticamente.
`autoinstall`	Conta com as opções `yes` ou `no`. Permite habilitar a instalação automática das atualizações de *kernel* que surgirem. Esta opção gera um *job* que será executado de hora em hora pelo `cron`.

Para os testes deste trabalho, optou-se por não utilizar *proxy* nem habilitar as atualizações automáticas. Somente foi habilitado a aplicação automática dos updates previamente instalados após o processo de reinicialização do Sistema Operadional. A Figura **??** mostra as opções utilizadas nos testes.

```
[Auth]
accesskey = 7bacac94562014d0b9f4d4c915823e2f6e61743def7cd6d826c252edff976fce

[Network]
https_proxy = None
gconf_proxy_lookup = no

[Settings]
install_on_reboot = yes
autoinstall =
```

Figura 3.1: Arquivo `/etc/uptrack/uptrack.conf` utilizado nos testes

As ferramentas instaladas e uma breve descrição de suas funções são exibidas na Tabela **??**.

Tabela 3.2: Aplicativos instalados

Aplicação	Descrição
`uptrack-install`	Permite a instalação de uma atualização específica a partir do recebimento de um `ID` como parâmetro.
`uptrack-manager`	Interface gráfica de administração do ksplice.
`uptrack-remove`	Permite a remoção de uma atualização aplicada. Se o arquivo `/etc/uptrack/disable` existir, a utilização deste comando não será permitida.
`uptrack-show`	Exibe os updates disponíveis e instalados (`-all`), os disponíveis para instalação (`-avaiable`) ou caso não seja passado parâmetro, exibe apenas os instalados.
`uptrack-upgrade`	Instala todas as atualizações disponíveis até o momento de sua execução.

3.3 Vulnerabilidade a Ser Explorada

A vulnerabilidade tratada nesta Seção será explorada através do uso de um *exploit*. Isso permitirá demonstrar o sucesso das correções aplicadas através da ferramenta Ksplice e também os benefícios que sua utilização pode trazer ao ambiente computacional.

Trata-se da falha nomeada como CVE-2010-2959[2] pela CVE. Ela está presente no código net/can/bcm.c, que trata do gerenciamento de *broadcast*.

Esta vulnerabilidade afeta as versões do *kernel* listadas na Tabela **??**.

Tabela 3.3: *Kernels* afetados pela vulnerabilidade CVE-2010-2959

Família	*kernels* **afetados**
`2.6.x`	Todos os anteriores ao 2.6.27.53
`2.6.32.x`	Anteriores ao 2.6.32.21
`2.6.34.x`	Anteriores ao 2.6.34.6
`2.6.35.x`	Anteriores ao 2.6.35.4

Esta vulnerabilidade permite a quem explorá-la a execução de código arbitrário, que será o caso deste trabalho, ou então causar uma negação de serviço através da geração de um tráfego excessivo.

O *exploit* para esta vunerabilidade, que causa o estouro de inteiro e permite a execução de código arbitrário, pode ser encontrado no site da SecurityFocus[3].

Neste *exploit* específico já está embutido a execução de um *payload*[4] que permitirá a obtenção de um *shell* com privilégios de *root*.

Para os testes que serão realizados no Capítulo 4, será necessário que este *exploit* esteja devidamente compilado. Para isto, é necessário efetuar o *download* do site da SecurityFocus, citado anteriormente, e a compilação do código utilizando o GCC.

[2] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2959

[3] http://www.securityfocus.com/data/vulnerabilities/exploits/42585.c

[4] Em protocolos de comunicação refere-se ao dado real que está sendo transmitido, neste caso, comandos arbitrários.

Capítulo 4

Testes e Resultados

4.1 Demonstração

A demonstração de utilização do Ksplice consistirá na exploração da vulnerabilidade antes da correção da falha através do Ksplice e após a aplicação do *patch* específico para correção desta falha, explorada pelo *exploit* citado na Seção **??**. Espera-se com isto, demonstrar a efetividade da correção sem a reinicialização.

4.2 Exploração da Vulnerabilidade Antes da Atualização

O *exploit* compilado na Seção **??**, permite a escalação de privilégios. Quando ele é executado por usuário com privilégios comuns, permite a obtenção de um *shell* com privilégios de administrador, o que é demonstrado na Figura 4.1.

Com privilégios de administrador seria possível, entre outras coisas: obtenção de quaisquer dados do *host* atacado, instalação de pacotes, criação de novas contas de usuário, etc. Enfim, as possibilidades são variadas e seu resultado final é o mesmo: comprometimento da segurança do *host* em questão.

4.3 Aplicação do *Patch*

Entre os inúmeros pacotes de atualizações disponíveis através do Ksplice, encontra-se a que corrige a falha proposta pela CVE-2010-2959, Figura 4.2.

```
islan@testeksplice:~/Download$ date; ./Exploit-CVE-2010-2959
Ter Abr  5 00:08:44 BRT 2011
[+] looking for symbols...
[+] resolved symbol commit_creds to 0xc016dcc0
[+] resolved symbol prepare_kernel_cred to 0xc016e000
[+] setting up exploit payload...
[+] creating PF_CAN socket...
[+] connecting PF_CAN socket...
[+] clearing out any active OPs via RX_DELETE...
[+] removing any active user-owned shmids...
[+] massaging kmalloc-96 SLUB cache with dummy allocations
[+] corrupting BCM OP with truncated allocation via RX_SETUP...
[+] mmap'ing truncated memory to short-circuit/EFAULT the memcpy_fromiovec...
[+] mmap'ed mapping of length 328 at 0xb783f000
[+] smashing adjacent shmid with dummy payload via malformed RX_SETUP...
[+] seeking out the smashed shmid_kernel...
[+] discovered our smashed shmid_kernel at shmid[285] = 27427123
[+] re-smashing the shmid_kernel with exploit payload...
[+] launching root shell!
root@testeksplice:~/Download# date
Ter Abr  5 00:08:48 BRT 2011
root@testeksplice:~/Download# whoami
root
root@testeksplice:~/Download#
```

Figura 4.1: Executando o *exploit* antes da atualização

Esta atualização específica pode ser instalada através do comando `uptrack-install -y <ID da correção>`, conforme mostra a Figura 4.3.

Neste trabalho, a atualização da falha do *kernel* utilizada nos teste e também para as anteriores à ela demorou aproximadamente 30 segundos, entre *download* e instalação. Porém, o processo de substituição das referências às funções envolvidas na atualização, que ocorre a partir do recurso stop_machine do Linux invocado pelo Ksplice, demora menos de 0,7 milesegundos. Este tempo é tolerado pelo Sistema Operacional e não causa nenhum prejuízo aos processos em execução no Sistema Operacional.

```
root@testeksplice:/home/islan# uptrack-show --all |grep 2959
[x9fv9ali] CVE-2010-2959: Privilege escalation in Controller Area Network subsystem.
root@testeksplice:/home/islan#
```

Figura 4.2: Verificação de correções para a vulnerabilidade CVE-2010-2959

Conforme pôde ser observado, inúmeras outras atualizações anteriores à solicitada foram instaladas automaticamente para manter a coerência da segurança no

```
root@testeksplice:/home/islan# date; uptrack-install -y x9fv9ali; date
Ter Abr  5 00:20:37 BRT 2011
The following steps will be taken:
Install [napqnyph] CVE-2010-1162: Memory leak in TTY layer
Install [dig0abud] CVE-2010-1488: Denial of service in proc_oom_score.
Install [w6eztmqd] CVE-2010-1148: NULL pointer dereference in CIFS
Install [kt4myh9t] CVE-2010-1146: Privilege escalation in ReiserFS
Install [i5jy1y8h] CVE-2009-4537: Remote buffer overflow in r8169 driver.
Install [e4ytxo1b] NULL pointer dereference in readahead with btrfs.
Install [vtd9uwxt] Denial of service in CIFS with remote OS/2 server.
Install [vu6taup2] Race condition in inotify watch addition/removal.
Install [32zxywjm] Resource leak in inotify release.
Install [z45tn6v6] NULL pointer dereference in EXT4_IOC_MOVE_EXT ioctl.
Install [pwc833k8] File corruption in ext4 due to wrong extent order.
Install [lwpvoaf5] CVE-2010-1173: Remote denial of service in SCTP.
Install [kn1ajufx] CVE-2010-1436: Denial of service writing GFS2 quota.
Install [ydhgrzxt] CVE-2010-1437: Denial of service in keyring subsytem.
Install [w5j3dujf] CVE-2010-1641: Insufficient privilege checking in GFS2 set_flags.
Install [eluf13i9] CVE-2010-2071: Privilege escalation in btrfs.
Install [gd0h0m8m] CVE-2010-2492: Privilege Escalation in eCryptfs.
Install [33rvof3x] CVE-2010-2803: Information leak in drm subsystem.
Install [x9fv9ali] CVE-2010-2959: Privilege escalation in Controller Area Network subsystem.
Installing [napqnyph] CVE-2010-1162: Memory leak in TTY layer
Installing [dig0abud] CVE-2010-1488: Denial of service in proc_oom_score.
Installing [w6eztmqd] CVE-2010-1148: NULL pointer dereference in CIFS
Installing [kt4myh9t] CVE-2010-1146: Privilege escalation in ReiserFS
Installing [i5jy1y8h] CVE-2009-4537: Remote buffer overflow in r8169 driver.
Installing [e4ytxo1b] NULL pointer dereference in readahead with btrfs.
Installing [vtd9uwxt] Denial of service in CIFS with remote OS/2 server.
Installing [vu6taup2] Race condition in inotify watch addition/removal.
Installing [32zxywjm] Resource leak in inotify release.
Installing [z45tn6v6] NULL pointer dereference in EXT4_IOC_MOVE_EXT ioctl.
Installing [pwc833k8] File corruption in ext4 due to wrong extent order.
Installing [lwpvoaf5] CVE-2010-1173: Remote denial of service in SCTP.
Installing [kn1ajufx] CVE-2010-1436: Denial of service writing GFS2 quota.
Installing [ydhgrzxt] CVE-2010-1437: Denial of service in keyring subsytem.
Installing [w5j3dujf] CVE-2010-1641: Insufficient privilege checking in GFS2 set_flags.
Installing [eluf13i9] CVE-2010-2071: Privilege escalation in btrfs.
Installing [gd0h0m8m] CVE-2010-2492: Privilege Escalation in eCryptfs.
Installing [33rvof3x] CVE-2010-2803: Information leak in drm subsystem.
Installing [x9fv9ali] CVE-2010-2959: Privilege escalation in Controller Area Network subsystem.
Ter Abr  5 00:21:09 BRT 2011
root@testeksplice:/home/islan#
```

Figura 4.3: Instalando o *patch* para a vulnerabilidade CVE-2010-2959

ambiente de testes. Uma listagem das correções instaladas pode ser obtida através do comando `uptrack-show` e sua saída pode ser vista na Figura 4.4.

4.4 Exploração da Vulnerabilidade Após Atualização

Após a instalação da correção, fez-se necessário a reexecução do *exploit* para a comprovação da resolução da falha. A rejeição da nova tentativa de execução é vista na Figura 4.5.

```
root@testeksplice:/home/islan# uptrack-show
Installed updates:
[napqnyph] CVE-2010-1162: Memory leak in TTY layer
[dig0abud] CVE-2010-1488: Denial of service in proc_oom_score.
[w6eztmqd] CVE-2010-1148: NULL pointer dereference in CIFS
[kt4myh9t] CVE-2010-1146: Privilege escalation in ReiserFS
[i5jy1y8h] CVE-2009-4537: Remote buffer overflow in r8169 driver.
[e4ytxo1b] NULL pointer dereference in readahead with btrfs.
[vtd9uwxt] Denial of service in CIFS with remote OS/2 server.
[vu6taup2] Race condition in inotify watch addition/removal.
[32zxywjm] Resource leak in inotify release.
[z45tn6v6] NULL pointer dereference in EXT4_IOC_MOVE_EXT ioctl.
[pwc833k8] File corruption in ext4 due to wrong extent order.
[lwpvoaf5] CVE-2010-1173: Remote denial of service in SCTP.
[kn1ajufx] CVE-2010-1436: Denial of service writing GFS2 quota.
[ydhgrzxt] CVE-2010-1437: Denial of service in keyring subsytem.
[w5j3dujf] CVE-2010-1641: Insufficient privilege checking in GFS2 set_flags.
[eluf13i9] CVE-2010-2071: Privilege escalation in btrfs.
[gd0h0m8m] CVE-2010-2492: Privilege Escalation in eCryptfs.
[33rvof3x] CVE-2010-2803: Information leak in drm subsystem.
[x9fv9ali] CVE-2010-2959: Privilege escalation in Controller Area Network subsystem.
root@testeksplice:/home/islan#
```

Figura 4.4: Atualizações instaladas

```
islan@testeksplice:~/Download$ date; ./Exploit-CVE-2010-2959
Ter Abr  5 00:47:49 BRT 2011
[+] looking for symbols...
[+] resolved symbol commit_creds to 0xc016dcc0
[+] resolved symbol prepare_kernel_cred to 0xc016e000
[+] setting up exploit payload...
[+] creating PF_CAN socket...
[+] connecting PF_CAN socket...
[+] clearing out any active OPs via RX_DELETE...
[+] removing any active user-owned shmids...
[+] massaging kmalloc-96 SLUB cache with dummy allocations
[+] corrupting BCM OP with truncated allocation via RX_SETUP...
[-] kernel rejected malformed CAN header
islan@testeksplice:~/Download$
```

Figura 4.5: Execução do *exploit* após a atualização de Segurança

4.5 Resultados e Discussão

Os testes realizados no Capítulo 4 focaram na eficácia do Ksplice como corretor de falhas no *kernel*. Esta eficácia foi demonstrada pelo teste em que foi executado um *exploit* para a vulnerabilidade CVE-2010-2959 antes e depois da aplicação da correção do *kernel* que corrigia esta falha através da ferramenta Ksplice.

O Ksplice vai além, podendo ser considerado também eficiente, pois cumpre seu papel como corretor de falhas acrescentando outros benefícios que os métodos antigos de atualização de *kernel* não possuiam. Entre os principais benefícios, pode-se citar alguns: facilidade de instalação, facilidade de configuração, comandos instalados com funções claramente dedutíveis, baixo nível de requisitos necessários para a utilização do *software*, pacotes de atualização extremamente pequenos, baixo consumo de conexão durante a atualização e finalmente a rapidez na aplicação das atualizações.

Além de sua eficácia e eficiência, o Ksplice tem acompanhado as tendências de mercado, suportando a maior parte das soluções de virtualização existentes. Isto garante às soluções de virtualização, que tanto o *host* hospedeiro quanto os *hosts* visitantes ou *guests* não estão vulneráveis à ataques que explorem vulnerabilidades de *kernel*.

Diante de todas estas características e dos testes demonstrados, o uso do aplicativo torna-se totalmente viável, principalmente em ambientes que necessitam manter a segurança sem comprometer a disponibilidade.

Porém, deve ficar claro que esta solução é apenas mais uma das tecnologias disponíveis para incrementar a segurança e não deve ser considerada como solução para todos os riscos que um ambiente pode apresentar. Afinal, de nada adiantaria o Sistema Operacional de um servidor estar sem nenhuma vulnerabilidade e o servidor *web* executado sobre este Sistema Operacional estar completamente desatualizado e consequentemente cheio de falhas de segurança. A porta de entrada continuaria existindo.

Capítulo 5

Conclusão

Sabe-se que um ambiente 100% seguro é utopia. Há sim ambientes que implementam o máximo de segurança possível para uma determinada época. Isso significa que o ambiente considerado seguro lança mão do máximo de ferramentas e tecnologias disponíveis até aquele momento para garantirem o maior nível de segurança possível.

Neste caso, o Ksplice é uma das ferramentas que contribuem para o incremento da segurança. Porém, utilizar somente ele não garante o *status* de ambiente seguro, já que ele trata somente as falhas presentes no *kernel*. Portanto, uma política de aplicação de correções de segurança dos outros aplicativos que rodam sobre o Sistema Operacional é essencial para a redução de possibilidades de comprometimento deste ambiente.

Diante dos benefícios citados no Capítulo 4, pode-se afirmar ser vantajoso a adoção do Ksplice em ambientes que utilizam o GNU/Linux, principalmente em ambientes onde a disponibilidade é fator crítico. Afinal, quanto maior a criticidade de um ambiente menor será o tempo disponíveis para manutenções programadas neste ambiente.

Prova desta afirmativa são os inúmeros clientes que a empresa possui atualmente, entre eles grandes *players* de mercado no fornecimento de servidores dedicados (VPS - *Virtual Private Server*) de alta disponibilidade e também outros grandes clientes, como a Canonical (mantenedora da distribuição Ubuntu) e a própria Intel.

5.1 Proposta Futura

Conforme dito na Seção 2.6, a empresa disponibiliza duas versões do Ksplice com a justificativa de que a versão paga oferecerá pacotes com módulos de atualização previamente analisados e testados por eles, já que nem todos *patches* podem ser diretamente fornecidos à aplicação. Há casos em que os *patches* fazem alterações nas estruturas persistentes do kernel, e neste caso há a necessidade de reescrita do código deste *patch de kernel*.

Diante deste cenário, seria interessante saber identificar quais *patches* podem ou não ser utilizados diretamente na ferramenta *open source* oferecida pela empresa desenvolvedora. Além disso, pode-se demonstrar a aplicação de um *patch* original utilizando o Ksplice na versão aberta.

Referências Bibliográficas

AHO, A. V.; SETHI, R.; ULLMAN, J. D. *Compiladores – Princípios, Técnicas e Ferramentas*. [S.l.]: Editora Guanabara Koogan S.A., 1995. Tradução Daniel de Ariosto Pinto.

ARNOLD, J.; KAASHOEK, M. F. Ksplice: automatic rebootless kernel updates. In: *Proceedings of the 4th ACM European conference on Computer systems*. New York, NY, USA: ACM, 2009. (EuroSys '09), p. 187–198. ISBN 978-1-60558-482-9. Disponível em: <http://doi.acm.org/10.1145/1519065.1519085>.

CORBET, J.; KROAH-HARTMAN, G.; RUBINI, A. *Linux Device Drivers*. Terceira edição. [S.l.: s.n.], 2005. ISBN 0-596-00590-3.

HESS, P. Coluna do pablo. *Linux Magazine*, n. 62, p. 22, 2010. ISSN 1806-9428.

KSPLICE INC. *Blog da Ksplice Inc.* 2011. Acessado em 12 de abril de 2011. Disponível em: <http://blog.ksplice.com/page/4/>.

KSPLICE INC. *Site da Ksplice Inc.* 2011. Acessado em 10 de abril de 2011. Disponível em: <http://www.ksplice.com/software>.

LOVE, R. *Linux Kernel Development*. 3rd. ed. [S.l.]: Addison-Wesley Professional, 2010. ISBN 9780672329463.

NAKAMURA, E. T.; GEUS, P. L. de. *Segurança de redes em ambientes cooperativos*. [S.l.]: Editora Futura, 2003. ISBN 85-7413-179-2.

PEREIRA, M. R. *Kernel do Linux*. Lavras-MG: UFLA/FAEPE, 2006.

RED HAT INC. *Red Hat Enterprise Linux 6 - Virtualization Guide*. 1. ed. [S.l.], out. 2010.

SECUNIA. *Vulnerabilidades do kernel 2.6.x de 2003 -2011*. 2011. Acessado em 10 de abril de 2011. Disponível em: <http://secunia.com/advisories/product-/2719/?task=statistics>.

SOUZA, A. D. Gerenciamento de nível de serviços: como tirar o melhor proveito. 2007. Disponível em: <http://www.hdo.com.br/v1/ideias/artigos/artigo22.html>.

TIBET, C. V. *Linux - Administração e Suporte*. [S.l.]: Editora Novatec, 2001. ISBN 85-85184-95-7.