**Ataliba de Oliveira Teixeira**

# Uma visão forense dos RootKits em Sistemas Linux

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. Sandro Pereira Melo

Lavras
Minas Gerais - Brasil
2005

**Ataliba de Oliveira Teixeira**

# Uma visão forense dos RootKits em Sistemas Linux

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 *11 de dezembro de 2005*

Prof. Joaquim Quinteiro Uchôa

Prof. Herlon Ayres de Camargo

Prof. Sandro Pereira Melo
(Orientador)

Lavras
Minas Gerais - Brasil

# Agradecimentos


- Minha noiva, Alana, pela paciência e apoio.
- A meus pais, pois sem eles nada disto seria possível.
- Aos Prof. Alexandre Dezem Bertozzi e Igor Prata pela inspiração.
- Meus colegas de trabalho Mario Machado, Marcus Jentzch, Francisco Américo, Weverton ( Baiano ), Rodrigo Ziviani e Luciana pelo apoio e conselhos na elaboração do trabalho, além da paciência de me ouvir falando da monografia quase todos os dias.
- Meu antigo chefe e amigo Alessandro Zebral pelo apoio e sempre compreensão.
- Aos meus amigos, que mesmo a distância participam das minhas conquistas.
- Fernando Fortes, Guilherme Veloso e Marcelo Leal, grandes amigos que fiz nesta pós-graduação.
- Ao meu orientador, Sandro Melo, pela paciência em receber os diversos emails com as versões dos capítulos e ainda, pelas dicas.
- Aos membros da banca, Herlon Aires Camargo e Joaquim Quinteiro Uchôa, pelas críticas e dicas que ajudaram em muito na melhora da qualidade deste trabalho
- E a todos, que direta ou indiretamente forneceram alguma ajuda para que este trabalho fosse feito.

# Resumo

Este trabalho tem apresenta um discussão detalhada sobre a investigação forense de intrusões em sistemas computacionais, tendo como objetivo fornecer subsídios para o melhor entendimento das taxonomias envolvidas em uma invasão utilizando *Rootkits*.

*Dedico este trabalho a minha noiva, Alana, a meus pais e meus irmãos.*

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

A internet é de fácil acesso para qualquer indivíduo munido de um computador e uma conexão de rede ( ou linha telefônica ). Pessoas físicas e jurídicas, em todo o mundo, podem achar qualquer ponto da rede sem se preocupar com nacionalidade, localização geográfica ou fuso horário. Entretanto, todas essas facilidades ao acesso da informação trazem riscos intrínsecos a seus métodos. Há o risco das informações serem perdidas, roubadas, corrompidas ou misturadas, além do sistema computacional poder vir a ser corrompido.

Quando a informação é gravada eletrônicamente e é disponibilizada via rede, ela torna-se mais vulnerável do que se a mesma for impressa no papel e trancada em um cofre.

Intrusos não precisam entrar no escritório ou em casa, e nem ao menos precisam estar no mesmo país. Eles podem roubar ou alterar informações sem mesmo tocar num pedaço de papel ou usar uma foto-copiadora. Eles podem criar novos arquivos, rodar seus próprios programas, além de ocultar evidências de suas atividades ilícitas.

E isto se faz cada vez mais problemático, pelo papel da informação em nossa atual sociedade. Vivemos o que é chamado por vários teóricos e pesquisadores como a *Era de Informação* ou *Segunda Revolução Industrial*. Isto acontece devido a uma profunda e contínua evolução tecnológica. Isto se deu principalmente por causa da microeletrônica e a disseminação dos computadores em lares no mundo inteiro (UCHôA, 2004).

O atual período histórico apresenta grandes modificações nos contextos políticos, sociais e econômicos, além dos culturais. Grande parte destas modificações tiveram origem nos avanços nas áreas de informática, microeletrônica e telecomunicações, que modificaram os relacionamentos humanos através de novas tecnologias de Informação e Comunicação (UCHôA, 2004).

Neste contexto, a informação passou a ser um ativo de importância permantemente crescente, similar a do capital. No mundo dos negócios, onde a informação sempre teve um papel muito importante na tomada de decisões, os processos se tornaram mais críticos. E a *TI*, a *Tecnologia da Informação* passou a exercer um papel cada vez mais estratégico.

Este valor agregado ao produto de *TI* e a informação gerada pelo mesmo, acabaram por criar um novo conjunto de problemas, ligados ao valor agora inputado aos dados guardados em uma empresa e/ou micro ligado a internet.

Neste novo mundo da informação, toda as relações foram transformadas, entre elas o crime e a forma com que as pessoas se relacionam com os mesmos.

Um novo tipo de crime teve origem em nossa época, o crime digital, que pode ser definido como aquele que faz uso de recursos computacionais e de telecomunicações para alcançar seu fim.

Os criminosos digitais se utilizam de diversas ferramentas para que alcancem o seu fim, que é o comprometimento de um sistema computacional.

Em um mundo onde a *TI* alcançou o *status* de produto, ou até patrimônio, a defesa das informações passou a ser uma preocupação.

Os criminosos digitais, iniciaram o desenvolvimento de diversas ferramentas que pudessem ajudá-los a alcançar seu fim, que é o comprometimento de sistemas computacionais.

Com o passar dos anos, as técnicas utilizadas por estes criminosos, levaram ao desenvolvimento de uma vertente na Ciência da Computação. Esta vertente, hoje muito conhecida, é a Segurança Computacional.

A Ciência da Computação caracteriza-se por ser o estudo de algoritmos e suas aplicações e sua aplicação para resolução de problemas em sistemas computacionais(VáRIOS, 2005).

Assim, a Segurança Computacional é caracterizada pelo estudo de técnicas que visam fornecer um estado de proteção ao patrimônio computacional e intelectual de uma organização.

A segurança de um sistema computacional está ligada a quatro pontos básicos, a confidencialidade, a disponibilidade, a confiança e a integridade das informações em um sistema computacional.

Para que o administrador de redes, possa desenvolver técnicas para defender sua rede, cumprindo os pontos básicos, um dos métodos é conhecer as ferramentas utilizadas pelos criminosos digitais.

É diante deste cenário, que a motivação para este trabalho surgiu. Os *rootkits* são as ferramentas mais utilizadas no comprometimento de sistemas computacionais.

Para que tal trabalho fosse realizado, foi necessário aplicar os conhecimentos de uma outra área da Computação, a Computação Forense.

A computação forense está relacionada ao trabalho de investigação, em sistemas e/ou redes de computadores, em busca de evidências criminais que possam ser levadas a um tribunal e que permitam reconstruir toda a seqüência de eventos de um ato criminoso, usando computador. Isto permite, entre outras coisas, que falhas de segurança sejam corrigidas e atacantes identificados.

O trabalho de um perito forense, em análises de cenários de invasão, é comparável a um médico legista. A autópsia é feita para verificar as causas da morte de um sistema, ou seja, quais vulnerabilidades foram exploradas para comprometê-lo e quais métodos foram aplicados para tal fim.

As mudanças efetudas pelos *rootkits* em sistemas computacionais, devem ser uma preocupação em perícias envolvendo os mesmos. Assim, o perito forense, não deve se basear somente no atual estado do sistema, mas, também, na recuperação de informações que possam asserir melhor qualidades as evidências encontradas.

Para isto, métodos devem ser aplicados para recuperação das informações em um sistema computacional comprometido. Uma segunda motivação deste trabalho, foi a construção de um método de perícia forense, tendo como foco principal as invasões utilizando *rootkits*.

Para este fim, foram utilizadas técnicas, que formalizam um método para análises gerais de taxonomias de invasão.

O método aplicado, por ter se mostrado funcional, se torna uma fonte importante de documentação para administradores de rede, interessados em aprender mais sobre como defender suas redes de pessoas mal intencionadas.

O trabalho se divide em quatro capítulos.

No segundo capítulo é apresentado um panorama geral de como o tema é tratado atualmente.

No terceiro capítulo é apresentado o modo como foram levantados as mudanças efetuadas por um *rootkit* em um sistema operacional *Linux*. Para este fim, foram utilizadas diversas ferramentas, em especial o *snoopy*[1], que possui a função de mapear todas as chamadas da função execve() do *Kernel do Linux* facilitando o levantamento das características de cada *rootkit*.

O quarto capítulo apresenta as propostas de trabalhos futuros, bem como as conclusões sobre o tema apresentado.

[1] http://packages.debian.org/unstable/admin/snoopy

# Capítulo 2

# Revisão Bibliográfica

## 2.1 Comentários Iniciais

“O que foi feito?”, “Quando foi feito?”, “Como foi feito?” e “Como se infiltrou?”, são algumas das perguntas que uma Perícia Computacional tem a intenção de responder. Através do processo de resposta a um incidente, o elemento humano, ou seja, o perito, através de seus procedimentos e investigações tenta levantar informações que possam levar a implicações contra um atacante, durante um processo judicial.

Estas informações coletadas durante este processo, podem ser chamadas de evidências. Evidências, segundo KRUSE , pode ser definida como qualquer informação que possua caráter probatório, descartando ou confirmando alguma teoria sobre a verdade dos fatos.

Estas evidências coletadas são a maior preocupação em todo um processo de análise forense. Se o perito não conseguir comprovar a autenticidade dos resultados obtidos durante uma análise forense, todo o processo pode ser refutado durante sua aplicação em um processo criminal. É recomendável que o perito documente todos os procedimentos adotados, e, ainda, garanta a integridade das informações.

A padronização é uma das armas neste processo. Uma metodologia padrão provê proteção às evidências através da definição de passos comuns que devem ser seguidos durante o processo de investigação. Caso os procedimentos utilizados na manipulação das evidências sigam todas as recomendações previstas nos padrões, a contestação da integridade se torna mais difícil.

Apesar do consenso geral em torno da necessidade de padronização, a definição de procedimentos para a forense computacional encontra dificuldades, uma vez que existem dezenas de tipos de mídia, sistemas operacionais com inúmeras versões, diferentes tipos de hardware e arquitetura. Além disso, o fato de que cada

caso possui circunstâncias praticamente únicas, dificulta sobremaneira a definição de procedimentos que cubram todos os possíveis aspectos de uma análise.

Com a padronização de métodos referentes a perícias computacionais, é possível construir uma base de conhecimento sobre as taxonomias de ataque mais comuns em sistemas computacionais.

E, com base nas taxonomias apresentadas nestes estudos, é possível formar possibilidades de melhores respostas aos incidentes de segurança, formatando, desde a sua instalação, os sistemas computacionais, para fornecer dados relevantes quando ocorrer alguma intrusão.

## 2.2 Rootkits

Para iniciar um processo de análise referente a *rootkits* é importante conhecer as peculiaridades desta ferramenta.

Os *rootkits* são assim chamados pois são um conjunto de ferramentas utilizadas para manter o acesso a um sistema comprometido. Como nos sistemas *Unix* e *Unix Like* o usuário root é aquele que possui o perfil de administração, o nome *rootkit* denomina uma ferramenta que mantém ou provém privilégios de administração a um invasor.

A instalação de um *rootkit* é normalmente o último passo de um ataque para conseguir o perfil de superusuário em um sistema *Unix*. Os *rootkits* podem ser instalados localmente, quanto remotamente, caso o intruso tenha acesso via *SSH*, *VNC*, *XDMCP* ou qualquer outra forma de acesso remoto à sua máquina.

Este capítulo apresentou metodologias que devem ser aplicadas na extração de evidências em uma máquina que sofreu invasão, com base nas literaturas atualmente apresentadas sobre o tema.

O estado dos *rootkits* com o uso destas metodologias fornece subsídios para que se descubram as modificações que os mesmos efetuam nos sistemas operacionais.

A partir das informações captadas, pode-se melhorar a segurança e entender mais profundamente o modo como estes *rootkits* podem ser combatidos, ou, usados como fonte de informações relevantes em uma perícia computacional.

Uma vez instalado, o *rootkit* vai modificar o comportamento da máquina alvo, mas não de modo que seja visível ao administrador. O processo do *rootkit* não irá aparecer em um comando de listagem dos processos ou o módulo carregado não irá aparecer quando executarmos o "`lsmod`" (BOVET; CESATI, 2003).

Estas ferramentas tiveram o seu auge durante o período de 1996 a 1999, quando os servidores de grande empresas começaram a se tornar cada vez mais frequentes na internet. Inicialmente, os sistema abertos foram o alvo, mas com o tempo, sis-

temas como os da *Microsoft*, começaram a possuir também seus root kits, como o *BackOrifice* e o *SubSeven*, além de outros(MARCELO, 2000).

De acordo com Nelson Murilo e Klaus Steding-Jersen, entende-se por um RootKit tradicional aquele que não é implementado em kernel space ( espaço de kernel ) sob a forma de um LKM ( Loadable Kernel Module ) (MURILO, 2001).

Tipicamente, este *rootkit* é composto de versões modificadas de comandos importantes do sistema ( `ls, ps, netstat, crontab` ). Os comandos substituídos podem omitir informações importantes ao administrador, tais como processos, conexões, arquivos e logs do invasor, de modo que sua presença não seja detectada.

Geralmente os *rootkits* podem conter também outros programas, que são extremamente úteis ao invasor :

### 2.2.1 LKM Rootkits

Os *rootkits* tradicionais trabalham em espaço de usuário, modificando programas do próprio sistema operacional. Uma segunda geração de *rootkits* foi apresentada na revista eletrônica Phrack, por Pragmatic, em 1999, onde eram apresentadas técnicas de como manipular as system calls do sistema operacional diretamente por módulos carregados dinamicamente.

Este documento, entitulado "(nearly)The Complete Linux Loadable Modules", foi apresentado a comunidade, e mostrava uma nova maneira de desenvolver *rootkits*, baseados agora em funções de baixo nível do sistema operacional (PRAGMATIC, 1999). Inicialmente, a publicação mostrava como corromper diversas *system calls* em ambiente Linux, que permitiam ocultar processos, modificar permissões, ocultar módulos do sistema, entre outras funcionalidades.

Este tipo de software representou um grande problema para diversos administradores, já que estes RootKits trabalhavam com os *LKMs*, que são pequenos programas que se relacionam com o kernel modificando funcionalidades do sistema operacional sem a necessidade de reinicialização ou até, uma recompilação completa do kernel.

São utilizados em vários sistemas Unix para carregar módulos que funcionam como uma interface entre o kernel e os dispositivos físicos do computador, de modo que não gerem grandes transtornos ao funcionamento do sistema como um todo (MAXWELL, 2000).

Um rootkit que trabalha em espaço de kernel pode funcionar de diversos modos, entre eles :

- *Execution Redirection ou Redirecionamento de execução* : esta ação se caracteriza quando um programa em modo de usuário é chamado e o kernel, com sua system call interceptada, redireciona a execução a outro programa.

Um exemplo seria um usuário chamando o "/bin/bash", o shell padrão do Linux e o kernel ao interceptar esta execução, redireciona para um programa qualquer, como um localizado no caminho "/var/tmp/.trojan/chamaleon". Este programa, se utilizado por um invasor, poderia possuir todas as funcionalidades de um shell padrão do *Linux*, acrescidos de funcionalidades de um *backdoor* para possibilitar o acesso de root.
Um grande problema encontrado na detecção desta técnica, se encontra na checagem de integridade de sistema de arquivos, pois, como o binário "/bin/-bash" se encontra intacto estes programas não encontrariam nada errado.

- *File Hiding ou ocultação de arquivos* : o perito só vê o que o atacante quer que ele veja. Ao invés de modificar arquivos como o comando `ls` ou `echo *`, o atacante manipula as system calls que acessam os arquivos, de modo que ocultem arquivos que o atacante não queira que sejam mostrados.

- *Module and symbol hiding ou ocultação de módulos e símbolos* : os LKMs podem ser ocultados interceptando syscalls ou acessando diferentes arquivos dentro do diretório /proc, ou manipulando as estruturas de kernel. A idéia é a de filtrar as informações de uma ataque ao perito ou administrador.

- *Process Hiding ou ocultação de processos* : usando métodos similares aos citados anteriormente, o kernel engana os comandos `ps` e `lsof` .
  Geralmente, você passa informações para a função `main()`, que é a primeira função executada em um programa escrito na linguagem C.
  Um argumento de linha de comando é a informação que segue o nome do programa em um *shell* de um sistema operacional *Linux*.
  Há dois argumentos especiais na função main `argc` e `argv`. O parâmetro `argc` contém o número de argumentos da linha de comando e é um inteiro. Ele é sempre pelo menos 1, pois o nome do programa é qualificado como um argumento.
  O parâmetro `argv` é um ponteiro para uma matriz de ponteiros de caractere. Cada ponteiro desta matriz aponta para um argumento fornecido na linha de comando.
  Na linguagem C, os elementos de uma matriz são acessados através de números, utilizando o formato `argv[1]`.
  O sistema operacional acessa o nome de um programa através da posição `argv[0]` de um programa.
  Um invasor, pode ocultar suas ações em um sistema, manipulando a posição 0 ( Figura 2.1 ) de um programa para que o mesmo mostre um nome diferente do original, quando os comandos de controle de processos forem executados(PELáEZ, 2004) (Figura 2.2 ).

- *Network Hiding ou oculptação de conexões* : neste caso o Rootkit modifica as saídas mostradas pelos programas netstat e lsof.

- *Sniffer Hiding ou ocultação de sniffers* : modifica as mensagens do kernel referentes ao uso do modo promíscuo em interfaces de rede quando um sniffer está em execução.

Um exemplo de LKM Rootkit é o exemplo de uma system call modificada que impede o usuário de criar diretórios.

Na Figura 2.3, é apresentado um exemplo que mostra o poder que um RootKit deste tipo pode ter e ainda os problemas que o mesmo pode causar em servidores. O processo de localização e neutralização destes rootkits é trabalho e exige que o perito tenha cuidado ao manipular o sistema invadido. As soluções iniciais são analisar os módulos carregados e a enumeração de processos no diretório /proc.

Infelizmente, pela sua dificuldade de detecção,ainda hoje, estas ferramentas são encontradas em sistemas na internet, causando problema a administradores que não implementaram uma política de segurança em seus sistemas computacionais.

### 2.2.2 O /dev/kmem é seu amigo

Em 2001, foi publicado um artigo na revista eletrônica Phrack, entitulado “Linux on-the-fly kernel patching without LKM” (DEVIC, 2001) (Sd e Devic, 2001 ), que seria a nova escola de root kits e que hoje ainda é um problema para a comunidade de segurança. Os autores apresentavam neste artigo uma ferramenta, de nome Suckit, que modificava as funções do kernel do Linux em tempo real, sem a necessidade de carregamento de nenhum módulo de kernel.

Esta ferramenta tornou-se o estado da arte, já que não havia como detectá-la mediante os processos já utilizados por programas de segurança, que localizavam e neutralizavam seus antecessores. Uma das medidas que foram tomadas para desenvolver uma maneira de detectar esta ferramenta foi a tentativa de auditar qualquer modificação em arquivos/diretórios do sistema, já que, ao criar os programas de execução do rootkit, o mesmo precisa colocar seus arquivos no sistema.

De acordo com Raul Silez (PELáEZ, 2004), o Suckit cria uma tabela particular de syscalls desviando o ponto de execução do kernel para a mesma. Em suma, o que este rootkit faz, de uma maneira mais complexa é criar uma tabela própria através de comparação de bytes das informações das system calls e com isso redirecionar o kernel para uma espécie de tabela própria de símbolos e reescrever o kmem.

```
/*
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */
 /* Salve este arquivo como â gcux_tool.c â , compile execute e confira a lista de proces
- Compile : gcc -o gcux_tool gcux_tool.c
- Execute : ./gcux_tool &
- Conferindo : ps -ef | grep gcux | grep -v grep | grep giac | grep -v grep
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])
 {
   char *p;
   char giac[5];

   strcpy(giac,"giac");

   for ( p = argv[0]; *p ; p++ )
      *p = 0;

   strcpy(argv[0], giac);

   /* Voce tem 60 segundos para ver que o comando ps vai reportar giac como o nome do proc
   sleep(60);
   return 0;

 }
```

**Figura 2.1:** Exemplo de manipulação do argumento argv[0]

```
bash-3.00$ gcc -o gcux gcux_tool.c
bash-3.00$ ./gcux &
[2] 4064
bash-3.00$ ps -ef | grep gcux | grep -v grep
bash-3.00$ ps -ef | grep giac | grep -v grep
ataliba   4064  3006  0 00:19 pts/3    00:00:00 giac
[1]-  Done                    ./gcux
bash-3.00$
```

**Figura 2.2:** Resultado da listagem de processos com o gcux

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/unistd.h>
#include <asm/uaccess.h>
#include <linux/sched.h>
#include <syscall.h>

extern void* sys_call_table[];
asmlinkage int (*original_call)(const char *path);

asmlinkage int m_syscall(const char *path)
{
  return 0;
}

int init_module(void)
{
  original_call=sys_call_table[SYS_mkdir];
  sys_call_table[SYS_mkdir]=m_syscall;
  return 0;
}

void cleanup_module(void)
{
  sys_call_table[SYS_mkdir]=original_call;
}
```

**Figura 2.3:** Exemplo de manipulação da função mkdir

## 2.3 Planejamento

O planejamento é uma das partes mais importantes para um trabalho de perícia. Tendo em mentes as peculiaridades de um *rootkit*, cria-se uma necessidade de métodos direcionados para cada tipo de *perícia computacional*.

Estes métodos, já devem ser de conhecimento do perito, antes mesmo do início de uma análise, fornecendo ao mesmo a possibilidade de aplicar a melhor abordagem para cada tipo de investigação, definindo a ordem e a prioridade das atividades que serão executadas.

Douglas Schweitzer(SCHWEITZER, 2003) apresenta um modelo, baseado em *checklists* que podem ser usados em análises genéricas.

Um *checklist* pode ser comparado a um *algoritmo*.

*Algoritmos* podem ser definidos como uma sequência de ações executáveis para a obtenção de uma solução para determinado tipo de problema(ZIVIANI, 1999).

Um *checklist* seria um roteiro genérico, com o melhor conjunto de passos para a execução de uma *análise forense*.

Este roteiro, pode ser utilizado pelo *perito forense* como um roteiro impresso, contendo cada passo a ser efetuado com um espaço para anotações.

Durante cada fase do processo, o perito efetua as anotações correspondentes, possuindo ao final, um documento que cobre toda a executação da perícia.

Isto facilita ao final de todo o trabalho, a execução de um relatório, a ser utilizado para apresentação dos resultados em juízo ou a uma diretoria de empresa a procura de respostas ao incidente ocorrido.

Um outro ponto interessante na utilização de um *checklist* é que ele fornece ao perito uma visão de todo o processo a ser efetuado. Assim, o mesmo pode escolher o melhor conjunto de ferramentas a utilizar para a criação de seu *toolkit*.

No processo de preparação para o início de uma perícia, é importante que se efetuem a criação de uma cópia do disco rígido, através de comandos como o `cp` ou o comando `dd` do *Linux* (Figura 2.4 ).

Um *toolkit* é um conjunto de ferramenta que o perito deve possuir em mãos durante o processo de análise ou até, na criação de sua estrutura computacional.

De posse deste conjunto de ferramentas personalizado o perito poderá efetuar análises mais facilmente, conseguindo em um período menor de tempo informações que possam ajudá-lo.

O *Linux* fornece dentro de sua própria estrutura um conjunto completo de ferramentas.

Em perícias comuns, as evidências são armazenadas em ambientes cuja entrada é restrita, onde são tiradas fotos e são feitas minuciosas descrições e relatórios do local onde o crime aconteceu. Estes ambientes, geralmente, são protegidos até a chegada dos peritos.

Nas perícias digitais, o uso dos *hashs* se faz muito importante. Um *hash* também chamado de *digest*, é uma espécie de "assinatura"ou "impressão digital"que representa o conteúdo de um fluxo de dados. Um hash pode ser comparado a um

selo de embalagem que indica claramente se a embalagem foi aberta ou violada (ANôNIMO, 2000).

```
bash-3.00# dd if=/dev/hda of=/dev/hdc bs=8192
```

**Figura 2.4:** Criação de um hd imagem usando o comando dd

Estes dois comandos efetuam uma cópia bit a bit do disco rígido, criando uma cópia fiel do ambiente comprometido, fornecendo ao perito a possibilidade de efetuar diversos testes sem que o ambiente original tenha sido modificado.

Neste ponto do trabalho que o *hash* se torna importante. Com ele o perito poderá validar se a sua cópia é idêntica ou não ao original.

A criação de uma estação forense, que é um local específico onde os resultados dos comandos são remetidos para posteriores análises.

Para este fim, o *netcat* provém uma interface de fácil utilização. Em um cenário, onde uma estação forense está preparada para receber as informações na sua porta 2222, o perito pode utilizar uma combinação dos comandos `tee` e `md5sum` para que as informações sejam recebidas e seu hash seja gerado a cada instante (ATíLIO, 2003).

Na Figura 2.5 o comando *tee* recebe um fluxo de dados e os guarda em um arquivo replicando os dados para a saída padrão. Os dados são recebidos pelo comando md5sum, que gera o *hash* de cada dado gravado no arquivo de nome *arquivo.md5*.

```
# nc -l -p 2222 | tee arquivo | md5sum -b > arquivo.md5
```

**Figura 2.5:** Exemplo de uso do comando netcat

## 2.4 Live Analysis

Em um primeiro contato com um sistema a ser periciado, o perito deve escolher o modo como o mesmo efetuará a análise.

O primeiro tipo de análise que pode ser efetuada em um sistema computacional, é a *Live Analysis*. A *Live Analysis* pode ser definida como aquela feita em um sistema, sem o desligamento abrupto do mesmo.

Esse tipo de análise é extremamente importante para a investigação, uma vez que é a única oportunidade de coletar uma série de informações que não estariam disponíveis caso o sistema fosse desligado.

Um dos grandes problema deste tipo de análise é que, na possibilidade de ainda haver um invasor na máquina, o controle da mesma é extremamente difícil ao perito. O atacante ao perceber a presença ou ação de um perito pode tentar apagar seus rastros.

É importante, ao perito, em um processo de análise deste tipo, copiar o conteúdo da memória principal do sistema. Na memória principal estão contidos as informações daquele momento específico do sistema computacional.

A captura destas informações em sistemas operacionais *Linux* pode ser feita utilizando o comando *dd*.

No *Linux* todos os dispositivos são entendidos como arquivos, característica herdada da linguagem C. Na linguagem C existe o conceito de fluxo. Seja qual for o dispositivo de entrada ou saída, o C vai enxergá-lo como um fluxo. Todos os dispositivos são similares em seu funcionamento e independentes do dispositivo ao qual estão associados (SCHILDT, 1997).

Assim, as mesmas funções podem ser utilizadas para o acesso ao disco rígido ou terminal de vídeo.

O dispositivo de memória física no *Linux* pode ser acessado via arquivos */dev/kmem* ou */dev/kcore*. Uma característica do arquivo */dev/kcore* é que ele possui o mesmo tamanho da memória física do sistema.

Através do comando dd ( Figura 2.6 ), pode-se efetuar uma cópia bit a bit da memória do sistema para posterior análise.

```
bash-3.00# dd if=/proc/kcore /tmp/kcore
1075472+0 registros de entrada
1075472+0 registros de saída
```

**Figura 2.6:** Cópia do dispositivo kcore com o comando dd

### 2.4.1 Listagem dos processos em uma Live Analysis

Em uma perícia digital, é importante que o perito recupere o estado do sistema operacional naquele momento. Apesar da cópia física da memória, fornecer informações do estado do sistema operacional naquele momento, é importante ao analista conseguir o maior número de informações sobre os processos que se encontram em execução naquele momento (ATíLIO, 2003).

Para a recuperação dos processos, o *Linux* fornece interfaces muito amigáveis ao perito e ao administrador.

Cada processo em um ambiente *Linux* é executado com privilégios específicos que terminam o que o mesmo vai poder utilizar de recursos da máquina. Um

invasor como bons conhecimentos, através de *rootkits*, poderá modificar o funcionamento de um programa ou serviço, fazendo com que o mesmo opere de maneira inesperada (PELáEZ, 2004).

Existem várias formas de se acessar as informações relacionadas aos processos em execução em um sistema operacional *Linux*.

Através do comando `uptime`, encontrado em qualquer instalação padrão do *Linux*, pode-se obter a informação sobre o período de tempo que a máquina está ligada. Além desta informação, é possível através deste comando, obter-se a média de carga da máquina nos últimos 1, 5 e 15 minutos ( Figura 2.7 ).

```
bash-3.00$ uptime
 01:07:08 up  5:37,  5 users,  load average: 0.01, 0.04, 0.07
```

**Figura 2.7:** Resultado do comando uptime

Apesar de fornecer algumas informações as mesmas podem não ter grande utilidade para o perito. Se a média de carga em uma máquina estiver muito alta, isto pode caracterizar ou um grande uso da mesma, ou um comprometimento e uso do sistema para chegar a outras locais.

Se o sistema reiniciou, isto pode ter sido uma falha de hardware ou até, uma ação do invasor para ocultar seus rastros durante o carregamento dos *scripts* de inicialização do sistema operacional.

Uma lista detalhada dos processos em execução, pode ser conseguida através do comando `ps`.

Esta listagem de processos pode ser uma fonte de evidências para o perito, caso o mesmo tenha outras saídas do comando `ps` para efetuar a comparação. Para isto, o administrador do sistema computacional, teria que se antecipar a uma invasão, e executar periódicas listagens dos processos em execução em um sistema computacional.

Caso o comando `ps` não forneça uma saída que forneça informações importantes para o perito, o mesmo pode fazer uso do comando `top`. O comando `top` ( Figura 2.8 ) fornece uma listagem dos processos que estão ocupando mais *CPU* naquele momento.

Assim, um processo que esteja utilizando muitos recursos de um sistema computacional, pode caracterizar a existência de um *rootkit*.

Além do comando `top`, o comando `lsof` pode ser muito útil, pois lista todos os arquivos abertos no momento da execução de um processo.

O diretório `/proc` contém todas as informações de hardware e de processos que estejam em execução naquele momento em uma máquina. Este diretório é um pseudo sistema de arquivos, que pode ser escrito somente pelo kernel.

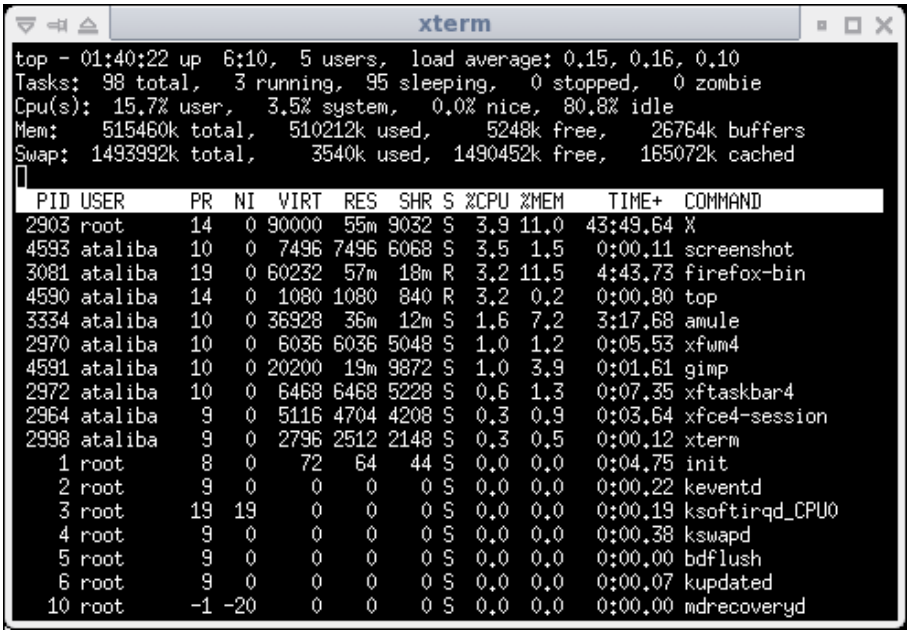


**Figura 2.8:** Exemplo de comando top

Cada processo[1], cria dentro do `proc` um diretório com o seu número *PID*. O *PID* ( *Process ID* ), que é o número de identificação do processo executado ( Figura 2.9 ).

Dentro do diretório criado pelo processo no `/proc`, há um arquivo chamado `cmdline`. O conteúdo do arquivo `cmdline` é a linha de comando completa utilizada para executá-lo. O comando `ps` mostra as informações contidas neste arquivo para mostrar o nome do mesmo na listagem de processos.

O conteúdo de `cmdline` pode se manipulado para não fornecer as informações corretamente ao perito, utilizando a técnica de manipulação do conteúdo de `argv[0]`.

Em conjunto com esta técnica, uma das taxonomias mais utilizadas de ataque por invasores é executar um programa e após isto, deletá-lo. No *Linux*, mesmo que tal ação tenha sido efetuada, o *binário* ( arquivo executável correspondente ao programa ) não será deletado até que o processo que o esteja utilizando seja terminado ou o sistema operacional desligado.

Para se detectar uma ação deste tipo, o perito pode se utilizar da interface fornecida pelo diretório `/proc`. No diretório do processo, há um arquivo de nome

[1]Processo é um programa em execução

```
bash-3.00$ ps ax | grep amule | grep -v grep
 3334 ?        R     12:21 amule
 3337 ?        S      0:00 amule
bash-3.00$ ls -la /proc/3334
total 0
dr-xr-xr-x    3 ataliba users 0 2006-01-07 11:33 .
dr-xr-xr-x  111 root    root  0 2006-01-06 17:29 ..
-r--r--r--    1 ataliba users 0 2006-01-07 11:33 cmdline
lrwxrwxrwx    1 ataliba users 0 2006-01-07 11:33 cwd -> /home/ataliba
-r--------    1 ataliba users 0 2006-01-07 11:33 environ
lrwxrwxrwx    1 ataliba users 0 2006-01-07 11:33 exe -> /usr/bin/amule
dr-x------    2 ataliba users 0 2006-01-07 11:33 fd
-r--r--r--    1 ataliba users 0 2006-01-07 11:33 maps
-rw-------    1 ataliba users 0 2006-01-07 11:33 mem
-r--r--r--    1 ataliba users 0 2006-01-07 11:33 mounts
lrwxrwxrwx    1 ataliba users 0 2006-01-07 11:33 root -> /
-r--r--r--    1 ataliba users 0 2006-01-07 11:33 stat
-r--r--r--    1 ataliba users 0 2006-01-07 11:33 statm
-r--r--r--    1 ataliba users 0 2006-01-07 11:33 status
bash-3.00$
```

**Figura 2.9:** Diretório padrão de um processo

`exe`, que nada mais é que um *link simbólico* para o binário do programa que foi executado (ATíLIO, 2003).

Links simbólicos são pequenos arquivos que apontam para um outro arquivo no sistema de arquivos. Na linha onde está o nome `exe` o arquivo estará marcado como `deleted`. Esta ocorrência caracteriza a taxonomia citada.

Para este tipo de busca, o perito deve utilizar os comandos `grep`, para procurar a string `deleted` e o comando `cp`, para copiar e efetuar posterior análise no *binário* coletado ( Figura 2.10 ).

Uma outra fonte de informação no diretório `/proc` é o sub-diretório `fd`, localizado dentro dos diretórios de cada processo. Este diretório contém informações sobre cada arquivo aberto por um processo ( incluindo bibliotecas ).

Na Figura 2.11 mostra os *file descriptors* ( descritores de cada arquivo ) dos arquivos abertos pelo processo `syslogd`, que tem o `PID` 66. Os *links* 0, 1, 2 referem-se respectivamente a entrada padrão, a saída padrão e saída de erro padrão.

O comando `lsof` pode ser uma interface de acesso ao diretório `fd`, pois o mesmo lista todos os arquivos abertos por um processo, tornando este processo menos penoso para o perito.

```
bash-3.00# /usr/bin/detectscand
bash-3.00# rm -f /usr/bin/detectscand
bash-3.00# ls -la /proc/* | grep deleted
lrwxrwxrwx    1 root root 0 2006-01-07 12:08 exe -> /usr/bin/detectscand (deleted)
bash-3.00# ps ax | grep detectscand | grep -v grep
 6715 pts/3    S      0:00 /usr/bin/detectscand
bash-3.00# cd /proc/6715
bash-3.00# cp exe /tmp/detectcand
bash-3.00# cd /tmp
bash-3.00# ls -la detectscand
-rwxr-xr-x  1 root root 48648 2006-01-07 12:09 detectscand
bash-3.00#
```

**Figura 2.10:** Procura pelos arquivos binários no sistema

```
bash-3.00# ls -la /proc/66/fd
total 0
dr-x------ 2 root root 0 2005-11-12 23:25 .
dr-xr-xr-x 3 root root 0 2005-11-12 23:25 ..
lrwx------ 1 root root 64 2005-11-12 23:25 0 -> socket:[75]
l-wx------ 1 root root 64 2005-11-12 23:25 1 -> /var/log/messages
l-wx------ 1 root root 64 2005-11-12 23:25 2 -> /var/log/syslog
l-wx------ 1 root root 64 2005-11-12 23:25 3 -> /var/log/debug
l-wx------ 1 root root 64 2005-11-12 23:25 4 -> /var/log/secure
l-wx------ 1 root root 64 2005-11-12 23:25 5 -> /var/log/cron
l-wx------ 1 root root 64 2005-11-12 23:25 6 -> /var/log/maillog
l-wx------ 1 root root 64 2005-11-12 23:25 7 -> /var/log/spooler
```

**Figura 2.11:** Conteúdo do diretório /proc/66/fd

### 2.4.2 Usuários Logados e informações de Login

A determinação de quais usuários estão logados no sistema pode fornecer informações importantes no momento da perícia. Isto pode ser determinado com o uso do comando `w`. O comando `w` lista, entre outras informações, os usuários logados, o endereço do sistema a partir de onde efetuaram o *login*, o horário do *login* e os utilitários que o mesmo está executando ( Figura 2.12 ).

Outra fonte importante de informações sobre os logins de usuário pode ser o comando `last`. O comando `last`, tal como o `w`, fornece informações se o usuário está logado ou não no sistema, local de onde o usuário logou ( no caso de login remoto ), data e tempo que o mesmo ficou conectado no sistema ( Figura 2.13 ).

```
$ w
 2:17PM  up 14 days, 13:18, 51 users, load averages: 0.27, 0.46, 0.47
USER    TTY -  LOGIN@  IDLE WHAT
yargo    p0 - 28Dec05  1:19 -ksh

hettich  p4 -  1:34AM     0 /usr/pkg/bin/pined -P /usr/pkg/etc/pine.conf
feanor   p5 -  1:25PM     2 /usr/pkg/bin/screen-4.0.2 -DD -RR
deanna   p7 - 28Dec05  8:11 irssi
krisu    p8 - 28Dec05  2days /usr/pkg/bin/pined -P /usr/pkg/etc/pine.conf
dno      p9 - Fri11PM  8:30 /usr/pkg/bin/screen-4.0.2 -R
catfive  pa - 28Dec05     6 emacs
kingjohn pb - Fri10PM  1:31 emacs
azmaveth pc - Thu09AM  2days irc
phm      pd - 12:18PM    26 -bash
rediske  pf -  5:21AM  8:46 /usr/pkg/bin/screen-4.0.2 -r
del      pg - 12:21PM    23 (pined)
avoyager ph -  2:07PM     6 /usr/pkg/bin/screen-4.0.2 -r
mahony   pi -  2:00AM  7:18 /usr/pkg/bin/pined -P /usr/pkg/etc/pine.conf
deanna   pj - Thu01PM 12:50 /usr/pkg/bin/zsh
fanthore pk - 10:15AM  4:00 /usr/pkg/bin/pined -P /usr/pkg/etc/pine.conf
feanor   pl -  1:25PM    51 /usr/pkg/bin/zsh
```

**Figura 2.12:** Saída do comando w

- nomes de usuários suspeitos ou usuários que não deveriam ter acesso de *login* a máquina.
- *logins* de origens suspeitas ( usuário local logando de outro país, exemplo ).
- *login* remoto de usuário não permitido

### 2.4.3 Conexões de Rede

Através da análise das conexões de rede e portas *TCP/UDP* em atividade, pode ser possível construir uma idéia do tipo de utilização que uma determinada máquina está fazendo da rede em um dado momento, assim como os processos da máquina.

Esta é a única oportunidade de coletar este tipo de informação volátil que não deixa nenhum tipo de rastro ou histórico depois que a conexão ou atividade é extinta, a não ser que cada aplicação cuide disso através da alimentação de arquivos de log, ou seja, utilizada alguma ferramenta específica para este fim, como o *TCP-Wrapper* comumente presente nas distribuições *Linux*.

```
bash-2.05$ last | more
ataliba   pts/3        0145184.wayinter Sat Jan  7 12:38   still logged in
oracleas  pts/3        201.50.77.2      Thu Jan  5 22:38 - 06:31  (07:53)
oracleas  pts/3        201.19.133.18    Thu Jan  5 20:10 - 20:13  (00:03)
oracleas  pts/3        201.19.133.18    Thu Jan  5 06:45 - 06:48  (00:03)
oracleas  pts/3        201.19.139.233   Wed Jan  4 19:38 - 20:18  (00:39)
jrocha    pts/3        alcmena          Wed Jan  4 14:35 - 15:06  (00:30)
ataliba   pts/3        ataliblnx        Wed Jan  4 14:35 - 14:35  (00:00)
ataliba   pts/3        ataliblnx        Wed Jan  4 14:32 - 14:34  (00:02)
oracleas  pts/6        201.19.139.233   Tue Jan  3 18:51 - 18:52  (00:01)
mario     pts/3        20150186039.user Tue Jan  3 18:37 - 20:26  (01:48)
oracleas  pts/3        201.19.155.92    Tue Jan  3 06:42 - 06:44  (00:02)
oracleas  pts/3        201.19.155.92    Tue Jan  3 06:37 - 06:39  (00:01)
ataliba   pts/3        0154069.wayinter Mon Jan  2 20:38 - 20:41  (00:03)
oracleas  pts/3        200165008117.use Mon Jan  2 18:27 - 20:24  (01:56)
ataliba   pts/3        0154069.wayinter Mon Jan  2 07:49 - 07:49  (00:00)
ataliba   pts/3        0154069.wayinter Sun Jan  1 19:15 - 19:16  (00:01)
oracleas  pts/3        201.19.254.67    Sun Jan  1 17:04 - 17:05  (00:00)
oracleas  pts/3        201008177100.use Sat Dec 31 14:42 - 14:45  (00:03)
```

**Figura 2.13:** Saída do comando last

A associação de serviços *Internet*, como *Telnet* e *HTTP* (*Hypertext Transfer Protocol*), com suas respectivas portas padrão, fornece uma idéia do tipo de atividade que gera determinado tipo de conexão, ou seja, a constatação da existência de uma conexão utilizando a porta 80 dá a idéia de que se trata de alguma aplicação fazendo uso do protocolo *HTTP*, como por exemplo um usuário utilizando um *browser*.

Note que a interpretação da atividade de rede de uma máquina pode não ser uma tarefa trivial, pois muitas vezes não é possível obter informações suficientes para que possam ser formuladas hipóteses viáveis.

Uma conexão utilizando a porta 80, por exemplo, tanto pode ser um navegador *Web*, quanto um *cavalo de Tróia* utilizando uma porta que geralmente tem seu tráfego permitido na maioria dos *firewalls*.

Sendo assim, existe a necessidade de interpretação dos dados que estão sendo transferidos, o que muitas vezes é impossibilitada pela utilização de canais seguros de comunicação como SSL (Secure Sockets Layer) ou SSH (Secure Shell).

Um programa que pode ser utilizado para capturar o tráfego de rede é o `tcpdump`. De acordo com Atílio, o `tcpdump` pode ser usado para capturar todo tipo de tráfego de rede, decodificar e exibir os datagramas a medida que eles são coletados

ou armazenar os *datagramas* em formato binário, permitindo análise posterior via `tcpdump` ou outros aplicativos (ATíLIO, 2003).

Na Figura 2.14, o `tcpdump` é usado para capturar e exibir os datagramas à medida que eles são coletados. A opção -l permite a visualização imediata da saída à medida que ela é produzida, a opção -n impede a conversão de endereços em nomes, a opção -e imprime o cabeçalho da camada de enlace, a opção -x imprime os datagramas no formato hexadecimal, a opção -vv imprime informações extras na saída do `tcpdump` e a opção -s determina o número de bytes de dados que deve ser capturado de cada datagrama (o padrão é 68 bytes).

No Figura 2.14, no segundo exemplo, o `tcpdump` é usado para armazenar os datagramas em um arquivo binário (*net.dump*), através da opção - w. Tal arquivo pode ser processado posteriormente, como no terceiro exemplo, através da opção -r.

```
# tcpdump -l -n -e -x -vv -s 1500
# tcpdump -w net.dump
# tcpdump -n -e -x -vv -s 1500 -r net.dump
```

**Figura 2.14:** Exemplos de uso do programa tpcpdump

Em primeira instância, o *sniffer* deve ser colocado em um ponto da rede onde ocorra maior tráfego da máquina invadida, ou também, pode se utilizar em *switches* gerenciáveis a utilização do recurso de *port mirroring*. Quanto maior o número de dados coletados, menor a possibilidade de que muitos datagramas sejam descartados durante o processo de análise (ATíLIO, 2003).

A análise dos datagramas capturados geralmente é feita com programas que reconstroem e exibem as informações em um formato mais adequado ao investigador. O `ethereal` permite a reprodução da sessão capturada, além da visualização dos eventos de uma maneira mais fácil do que é oferecida pelo `tcpdump`.

Além destas ferramentas, também há outras ferramentas no sistema operacional que podem ajudar no processo de análise.

- Arp: programa nativo que é capaz de exibir o cache de respostas obtidas pelo protocolo ARP. Através deste aplicativo é possível enumerar todos os endereços de enlace com os quais a máquina tem se comunicado no último minuto.

- Netstat: o objetivo desta ferramenta nativa é fornecer estatísticas de utilização da rede, mais especificamente dados sobre os protocolos *IP, UDP e TCP*. Ele exibe informações que podem ser úteis para o examinador, tais

como a lista de conexões ativas na máquina, as portas que estão aceitando conexões da rede e qual o estado atual da tabela de roteamento.

- Traceroute: indica quais roteadores estão entre a máquina local e um determinado destino em uma rede. Este aplicativo é nativo e pode ser útil caso sejam observadas rotas de rede suspeitas durante a utilização do netstat.

## 2.5 Post Mortem Analysis

O segundo tipo de perícia digital, é a análise *post mortem* de um sistema comprometido.

A análise *post mortem* é aquela feita após o sistema operacional ser desligado, seja por uma falha do sistema operacional ( ocasionada pela intrusão ) ou por um desligamento abrupto pelo administrador ou até, pelo perito.

É neste momento da análise forense, que ocorre uma das fases mais importantes do processo. A análise do sistema de arquivos. O disco rígido de um sistema computacional é uma fonte importante de evidência em um exame forense, pois diversos dados estão espalhados sobre toda a área do disco.

Alguns dados, não acessíveis via sistema operacional, também podem estar disponíveis. Quando deletado, a referência de um arquivo é removida das estruturas do sistema de arquivos, mas os dados continuam no disco. A remoção destes dados só ocorre, quando a área onde estavam é sobre escrita (SILBERSCHATZ; GALVIN, 2000) (WOODHULL, 2000).

Caso o perito ou o administrador tenham instalado o programa *LKCD*[2], que consiste em um *kit* de ferramentas que fornecem a condição de recuperar o estado do sistema antes de sua parada.

Isto acontece pois o *LKCD* guarda uma imagem do sistema que pode ser acessada via arquivo `/var/log/dump`.

Este *log* pode ser acessado via ferramenta `lcrash`, contida no *kit* do *LKCD*, que fornece prompt próprio onde podem ser observados, inclusive, os processos que estavam sendo executados em tempo de *boot*.

Para se utilizar esta função é necessário que o *kernel* tenha configurado a opção de *Sysreq*. Somente com o pedido de *Sysreq*, vindo diretamente do perito, é que a ferramenta `lcrash` será ativada e irá retornar um *prompt*, após a máquina reiniciar, para análise do *dump* da memória.

Esta ferramenta fornece uma possibilidade ao perito, de, no caso de um desligamento, ou um crash geral da máquina, conseguir analisar após o religamento da máquina, o momento anterior em que a mesma se encontrava(IBM, 2002).

[2]http://lkcd.sourceforge.net

Esta ferramenta se encontra no limiar de uma *Live Analysis* com a *Post Mortem Analysis*. Isto pode ser observado pois ela fornece um retrato do sistema anteriormente ao seu desligamento, para uma análise posterior.

### 2.5.1 Arquivos suspeitos, diretório e uso do disco

Um dos aspectos de uma invasão relacionada com *rootkits* é o uso de disco. O invasor irá gravar muitos arquivos em disco, referentes a sua invasão.

Por este motivo, a preocupação com a integridade do sistema de arquivos deve ser alta.

Um programa simples, que pode ser implementado facilmente em um sistema operacional com Linux, é o quick_lstat, que é um aplicativo básico de *IDS* ( *Intrusion Detection System* ) ou até para a manutenção de um sistema. Ele tem como objetivo detectar alterações nos arquivos do sistema Linux para identificar *trojans*, *cavalos de tróia* e fazer *checagem de sniffers*, catadores de senhas nos devices da rede.

O `quick_lstat` funciona do seguinte modo. Na sua primeira execução, o programa criará uma base de dados com os arquivos que foram selecionados no arquivo de configuração `/etc/lstat.conf`. Esta base de dados conterá as seguintes informações :

- Dono do arquivo ou pasta.
- Grupo do arquivo ou pasta.
- Permissões.
- Data da última alteração.
- O tamanho.
- Md5 do arquivo.

A partir destas informações armazenadas na base de dados é que será feita a verificação. A partir da primeira execução, toda vez que o programa for executado, ele irá fazer uma varredura do sistema, irá criar um arquivo temporário. Este arquivo temporário será comparado com a base de dados atual e caso haja alguma discrepância, o programa irá gerar um log com as modificações apresentadas.

Além disto, o programa checa se há a presença de algum *sniffer* no sistema operacional, gerando um *log* de aviso caso encontre algum ( Apêndice A ).

Tendo em vista de os *rootkits* tradicionais, modificam arquivos dentro do sistema operacional, o uso do quick_lstat se mostra interessante quando houver o

uso dos *logservers*, que são locais em uma rede de computadores, voltados para guardar os *logs* de todas as máquinas.

De acordo com Jarbas de Freitas Peixoto, o uso de um *logserver* é interessante, pois ele pode guardar externamente a máquina, dados que possam vir a ajudar em processos de auditoria. O *logserver* poderia portanto, facilitar o trabalho do perito, pois teria informações importantes sobre as mudanças ocorridas no sistema de arquivos em um período específico (PEIXOTO, 2002).

Para este fim, o programa quick_lstat pode ser configurado com a opção de *client/server*. Esta opção foi desenvolvida para que se possa centralizar os logs de integridade de sistemas operacionais em um logserver centralizado, separado de uma máquina que possa vir a sofrer uma intrusão. Este acesso aos logs do programa pode ser protegido por uma senha, configurada diretamente em tempo no arquivo de configuração `/etc/lstat.conf`.

Uma outra vantagem do `quick_lstat` é a possibilidade de personalização do programa. Tendo em vista que *crackers* possam ter acesso aos códigos do mesmo, pode-se alterar o nome dos arquivos que o programa gerencia. Após esta modificação dos dados é necessário que se recompile e modique-se algumas entradas no arquivo de configuração.

Com o uso do `quick_lstat` ou qualquer outro programa de checagem de integridade de sistema de arquivos, o processo de análise de uma intrusão se torna mais fácil, pois a comparação do estado inicial de um sistema de arquivos, com o estado pós invasão, pode mostrar os locais onde o cracker pode ter gravado ou guardado programas que possam facilitar suas ações.

Uma busca detalhada no sistema de arquivos também se faz importante. Um método, apresentado por Raul Silez , é o uso do `find` ( Figura 2.15 ). O `find` é uma ferramenta muito importante no processo de análise, por aceitar diversas configurações para encontrar arquivos. Uma série de variações deste comando pode levar o perito a encontrar evidências importantes que não seriam encontradas com outros métodos (PELáEZ, 2004).

```
Procurando por arquivos SETUID and SETGID :
# find / -perm +6000
- Procurando por arquivos que tenham permissão de escrita para todos:
# find / -perm -2
- Procurando por nomes que começam com "."
# find / '(' -name '.??*' -o -name '.[^.]' ')' -ls
- Procurando por arquivos que não tenham seu owner listado no /etc/passwd :
# find / '(' -nouser -o -nogroup ')' -ls
```

**Figura 2.15:** Exemplos de uso do find

No *Unix*, os diretórios ocultos iniciam com ".". Atacantes em geral colocam seus arquivos em diretórios nomeados como:

- “.. ” : dois pontos seguidos de um espaço vazio, que dificulta a detecção devido a parecer com o nome padrão do diretório local no *Linux*
- “. ” : ponto seguido de um espaço vazio, para dificultar a localização do diretório, já que o mesmo está oculto e sem um nome facilmente detectável.
- “...” : três pontos, que se parece em muito com o nome padrão do diretório atual dentro do *Linux*
- “ ” : palavras vazias, que não são detectadas pelo perito ou administrador em uma listagem rápida de um diretório.

### 2.5.2 Contagem dos hard links e total de blocos

Cada arquivo do *Linux* é associado um número para cada *hard link*. Se um arquivo no *Linux* é associado a um *hard link* o contador passa a ser 2 ( ele e o *hard link* ).

Em cada diretório do Linux há dois *hard links*, um correspondente ao próprio diretório ( `.` ) e outro correspondente ao diretório pai ( `..` ) .

Quando um atacante usa um *rootkit* de modo incorreto ( Figura 2.16, ele tenta esconder os sub-diretórios do perito ou administrador. Em uma situação em que um diretório contém sub-diretórios ( 3 sub-diretórios ) a contagem de hard-links deve ser 5, 2 associados ao padrão de cada diretório e 3 ligado aos diretórios contidos no mesmo.

Quando um atacante esconde arquivos no sistema, a contagem será diferente da listada via `ls`.

Um diretório padrão do *Linux* contém uma lista total de 8 blocos ( Figura 2.17 : 4 blocos relacionados ao diretório pai ( “.” ) e mais quatro blocos relacionados ao diretório atual ( “..”).

O comando `ls` do Linux lê o número total de blocos alocados pelo sistema, usando a unidade básica de alocação. Em sistemas de arquivos básicos do *Linux*, `ext2` e `ext3`, o tamanho do bloco é 4096 bytes. Esta informação pode ser checada usando o aplicativo `dump2fs` ( Figura 2.18.

Como uma conclusão, o número total de blocos mostrado pelo comando ls é um múltiplo de 4.

Quando um arquivo de diretório é criado , ele é associado a um tamanho menor ou igual a 4096 bytes ( +/- 4Kbytes ) . Se um arquivo tem 4097 bytes, ele acusa que o número de blocos é de 8 ( dois blocos de 4Kbytes ). Um bloco com zero bytes não aumenta o tamanho do diretório.

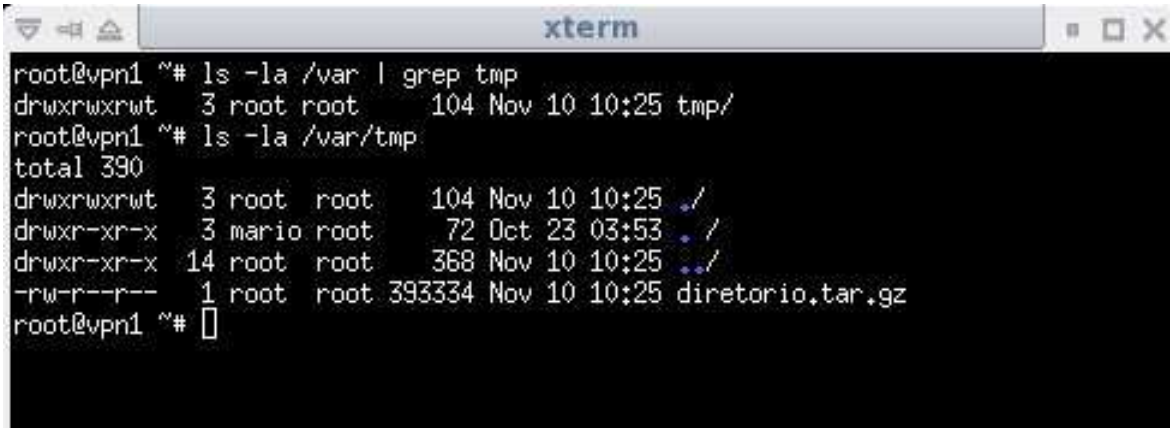


**Figura 2.16:** Exemplo de um diretório comprometido

```
root@vpn1 ~/teste# ls -al
total 8
drwxr-xr-x  2 root root  48 Nov 14 08:50 ./
drwxrwxrwx  8 root root 528 Nov 14 08:50 ../
```

**Figura 2.17:** Blocos de um diretório

```
bash-3.00# dumpe2fs /dev/hda5 | grep "Block size"
dumpe2fs 1.38 (30-Jun-2005)
Block size:               4096
```

**Figura 2.18:** Uso do comando dump2fs

A Figura 2.16 mostra o diretório /var/tmp o diretório contém um tamanho total de 390. Como o diretório a primeira vista se encontra vazio, conclui-se que há 382 bytes disponíveis que não estão alocados a nenhum diretório.

Além disto, a opção -s do comando `ls` pode permitir a contagem dos blocos em cada diretório ( Figura 2.19 ).

```
root@vpn1 /var/tmp# ls -sl
total 388
388 -rw-r--r--  1 root root 393334 Nov 10 10:25 diretorio.tar.gz
```

**Figura 2.19:** Opção -s do comando ls

### 2.5.3 MACTimes

Os *MACTIMES* podem fornecer informações importantes sobre um sistema comprometido. Deve-se ter em mente, que, apesar de serem fontes importantes em um processo de análise, são informações extremamente perigosas.

Um atacante pode adulterar os *MACTIMES* em disco. Ao substituir um arquivo, ele pode, via vários comandos do *Linux*, como o comando `touch`, retornar o mesmo para a data de gravação original. Assim, a informação que o perito terá via *MACTIME*, é que o arquivo não foi modificado, mas, ao contrário, o mesmo foi.

Por este motivo, é necessário usar com cuidado as informações fornecidas por esta opção de perícia, já, que, nem sempre elas realmente terão grande peso em uma análise forense.

Uma das mais básicas técnicas de verificação do sistema de arquivos é a de conferir os tamanhos e o `timestamp` de cada arquivo binário dentro do sistema operacional ( Figura 2.20. Todos os arquivos em um sistema Unix tem três tempos associados a ele : tempo de modificação ( modification time, mtime ) , tempo de acesso ( atime ) e change time ( ctime ).

```
Modification time (mtime, "ls -al") :
# ls -al my_file.txt
-rw-rw-r-- 1 root root 4 Apr 28 21:58 my_file.txt
- Change time (inode status) (ctime, "ls -alc"):
# ls -alc my_file.txt
-rw-rw-r-- 1 root root 4 Apr 28 22:01 my_file.txt
- Access time (atime, "ls -alu"):
# ls -alu my_file.txt
-rw-rw-r-- 1 root root 4 Apr 28 22:03 my_file.txt
```

**Figura 2.20:** Exemplos de como conferir os mactimes de arquivos em disco

O comando `stat` pode fornecer estas informações sem problemas, mas o TCT ( The Coroners ToolKit ) fornece uma ferramenta mais apropriada para este fim, o comando `mactime` ( Figura 2.21. O *TCT* é um conjunto de ferramentas voltadas para o exercício da perícia forense, em sistemas operacionais *Unix Like*.

```
bash-3.00$ stat top-3.5.1-sol9-sparc-local.gz
  File: ‘top-3.5.1-sol9-sparc-local.gz’
  Size: 83298           Blocks: 168        IO Block: 4096   regular file
Device: 307h/775d       Inode: 874         Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/ ataliba)   Gid: (  100/   users)
Access: 2005-11-04 09:47:39.000000000 -0200
Modify: 2005-11-04 09:45:03.000000000 -0200
Change: 2005-11-04 09:45:03.000000000 -0200
```

**Figura 2.21:** Exemplo de saída do comando stat

Um outro meio, utilizado para obter todos os timestamps do diretório para futura análise é o uso do comando `ls` com sua opção recursiva, `ls -R` ( Figura 2.22 ). Raul Silez apresenta três usos deste comando para captar informações para futura referência (PELáEZ, 2004).

```
# ls -alRu / > access_times.txt
# ls -alRc / > change_times.txt
# ls -alR / > modification_times.txt
```

**Figura 2.22:** Exemplos de uso do ls recursivo

# Capítulo 3

# Análise Comportamental de um Rootkit a partir de filtragem da System Call Execv

## 3.1 Considerações Iniciais

Um *rootkit* tem a função de modificar o comportamento de um sistema computacional, para torná-lo mais facilmente acessível ao invasor.

Para que o invasor tenha acesso aos seus dados em uma máquina comprometida, é necessário, que, pelo menos, o mesmo consiga gravar dados em disco.

Isto causaria modificações em áreas do disco e arquivos importantes do sistema computacional, que são importantes na análise comportamental de um *rootkit*.

Além disto, a filtragem da *system call* `execve()` se mostra importante, pois assim, é possível estudar todo o caminho efetuado por um *rootkit* ao comprometer um sistema computacional.

## 3.2 O Rootkit Adore-ng

O *Adore-ng* é um dos *rootkits* mais conhecidos para o sistema computacional *Linux*.

O *Adore-ng* tem seu funcionamento similar ao *rootkit Adore*, com a peculiaridade de subverter o *Linux* a partir de outro método. Ele substitui as funções de listagem de diretórios por rotinas próprias, provendo ao administrador as informações que o invasor quiser sobre o sistema de arquivos e o diretório `proc`(STEALTH, 2004b).

O *rootkit Adore-ng* foi criado por Stealth[1], membro do grupo de segurança *TESO*[2].

O *Adore-ng* foi implementado como um módulo de kernel ( *LKM* ) e manipula a camada de abstração do sistema de arquivos do *Linux*, chamada de *Virtual File System* ( *VFS* ). A razão principal para isto, é que o *rootkit* deve evitar o uso da `sys_call_table` para não ser detectado (PELáEZ, 2004) (STEALTH, 2004b).

Muitos programas de usuário obtém informações do diretório `proc`, que podem ter seus dados manipulados por um invasor. Entretanto, o *rootkit Adore-ng* usa a camada inferior para esta finalidade, o *VFS*, complementando ele com a manipulação do sistema de arquivos `/proc` para esconder as conexões *TCP* (STEALTH, 2004b) (PELáEZ, 2004).

As características padrão para esconder as ações do *rootkit* no sistema computacional são :

- `syslog` filtering : os logs gerados pelos processos que estão invisíveis não irão ser listados pelo `syslog`.
- `wtmp/utmp/lastlog` : nenhuma das entradas geradas pelos processos escondidos não irão aparecer no arquivo xtmp, exceto se isto for forçado usando processos invisíveis e autenticados.
- Possibilidade de utilizar a infecção de *LKMs* para que os mesmos sejam carregados no *boot* do sistema computacional.

A camada que trabalha em *user-mode* é implementada a partir da aplicação `ava`. É um software auto-explicativo a partir de suas opções ( Figura 3.1 ) (STEALTH, 2004b). Na Figura 3.2, é possível visualizar os comandos disponibilizados pelo aplicativo `ava`.

Entretanto, uma das funcionalidades principais do *Adore-ng* em relação ao *Adore* é que o mesmo pode ser controlado sem o uso do aplicativo `ava`.

Stealth ainda cita na revista eletrônica Phrack[3](STEALTH, 2003) algumas idéias sobre a evolução dos *rootkits de kernel*. Ele inicia o artigo citando algumas das funcionalidades de segurança que os *rootkits* em breve precisarão possuir (PELáEZ, 2004).

- Novas versões de kernel e extensões para as customizações de cada distribuição.

[1] http://stealth.7350.org
[2] http://www.team-teso.net
[3] http://www.phrack.org

```
# ava
Usage : ava {h,u,r,R,i,v,U} [file or PID]

                I print info (secret UID etc)
                h hide file
                u unhid file
                r execute as root
                R remove PID forever
                U uninstall adore
                i make PID invisible
                v make PID visible
```

**Figura 3.1:** Opções do aplicativo ava

```
# echo > /proc/<ADORE_KEY> cria o shell autenticado.
# cat /proc/hid_<PID> em que um shell torna invisível o PID.
# cat /proc/unhide_<PID> torna o processo novamente visível.
# cat /proc/uninstall desinstala o adore.
```

**Figura 3.2:** Opções do aplicativo ava

- Ausência de símbolos importantes ( *sys_call_table).*
- Mecanismos avançados de auditoria de *logs* e *logging*.
- Aumento de segurança no *kernel* ( *Hardening Kernel* ).
- Detecção de intrusos/detecção de comportamento anormal.
- Ferramentas avançadas de forense e métodos de análise.

Uma das funções mais importantes dos *rootkits* do futuro é tornar o *backdoor* invisível sempre que possível. Portanto, uma solução de eliminação de *logs* deve estar presente, e sua implementação se baseia em técnicas já apresentadas, como a subversão do *VFS*, impossibilitando a escrita dos *logs* em disco.

Além do *logging*, o *backdoor* deve estar disponível para conexão via rede escutando em uma porta específica. Os novos métodos se baseiam na modificação do sub-sistema de redes do *kernel*.

Todas estas novas idéias de implementação de *backdoors* são baseadas no *network kernel hooks* (BIOFORGE, 2003), redirecionamento de *VFS* (STEALTH, 2004a) e a técnica de obfuscação baseada na infecção um módulo existente do *kernel* (TRUFF, 2003).

A manipulação do sub-sistema de redes do *Linux*, é feita através dos *hooks* de kernel. *Hooks* podem ser comparados a gatilhos para certos eventos (PELáEZ, 2004).

O *kernel* do *Linux* possui estruturas pré-definidas para o tratamento do sub-sistema de redes, sendo inclusive tais estruturas utilizadas pelo comando *iptables*, que utiliza os *network kernel hooks* (KLAN, 2003).

Em alguns casos os *rootkits* substituem as *syscalls* referentes de rede, para que ele possa modificar o cabeçalho do pacote pelo seu próprio. Se o pacote possuir o cabeçalho modificado, é executado, se não, é executado o cabeçalho original.

Além disto, o redirecionamento de conexões também é muito utilizado. Um acesso a porta `80` do sistema computacional pode ser redirecionada para um *backdoor* que esteja trabalhando na porta `9999` do servidor.

Assim, a conexão não seria previamente notada pelo administrador, pois estaria ocorrendo em uma porta padrão e seu tráfego não seria estranho.

O grande trunfo do *Adore-ng* é a sua manipulação do *VFS* do sistema operacional *Linux*. *Virtual FileSystem* corresponde à camada de abstração entre o *FileSystem* e os aplicativos de usuários, ou seja, forma uma camada que permite que chamadas de sistema relacionadas a arquivos, sejam utilizadas independentemente do sistema de arquivos utilizado.

No mundo *Unix* desde as conexões de rede até os dispositivos de hardware, são tratados como arquivos. Assim, todos eles são tratados pela camada de abstração que é o *VFS*.

Assim, o *Adore-ng* consegue ocultar sua presença, já que trabalha diretamente na camada que trabalha com os arquivos. Um arquivo tornado invisível, não vai ser mostrado pelo comando `ls`, pois o próprio *kernel* já estará filtrando este resultado.

O administrador, assim, ficaria sem armas pois não encontraria nenhum resultado aparente, caso não houvessem ferramentas instaladas no sistema para mostrar-lhe o resultado.

Um outro ponto importante do *Adore-ng* é a obfuscação de *LKMs*. O comando `lsmod` quando executado mostra todos os módulos que estão carregados no sistema em um dado momento.

Isto acontece devido a uma manipulação da função `module_list()`, que cria uma lista dos módulos carregados no sistema. O *rootkit* manipula esta lista filtrando os resultado de não permitindo que alguns módulos sejam mostrados, quando o sistema acessa o arquivo `/proc/modules` que contém as lista dos módulos do sistema.

Adicionalmente, o módulo tenta não exportar nenhum símbolo ( usando a macro `EXPORT_NO_SYMBOLS` ) tornando-se invisível e funcional no sistema.

Assim, mesmo que haja uma suspeita, o administrador não conseguirá ver os módulos de um *rootkit* carregados, em especial, do *rootkit Adore-ng*.

O *rootkit Adore-ng* é uma ferramenta que pode causar muitos problemas a administradores e peritos, já que uma invasão bem construída, poderia obfuscar todas as ações do *rootkit* pois o mesmo possui métodos para este fim.

## 3.3 Ferramentas utilizadas na criação do ambiente de análise

O ambiente de análise forense proposto neste trabalho, consiste de duas partes distintas. Uma consiste das ferramentas *AIDE* e *Snoopy Logger*, ambas utilizadas para preparar o sistema operacional para gerar dados relevantes para a detecção de uma intrusão em um sistema computacional.

A outra consiste na escolha de ferramentas para formar um *Toolkit* capaz de fornecer ao perito uma plataforma de trabalho amigável e útil, para que o trabalho chegue a resultados realmente aproveitáveis.

Por este motivo, nesta seção, cada ferramenta utilizada na detecção das taxonomias do *rootkit Adore-ng* será analisada para demonstrar sua importância no processo.

## 3.4 Ferramentas Pré-Intrusão

### 3.4.1 Quick_lstat e Aide

Um dos aspectos de uma intrusão relacionada com *rootkits* é o uso do disco e modificações efetuadas no mesmo. Por este motivo, a preocupação com a integridade dos dados no disco deve ser grande, já que qualquer ação com tais programas pode causar modificações desastrosas no disco (ATíLIO, 2003) (SCHWEITZER, 2003).

Uma das formas de efetuar um estudo e catalogar as modificações efetuadas em um disco é o uso de um *IDS* relacionado ao sistema de arquivos do sistema computacional.

Um *IDS* deste tipo é importante e indicado para os diretórios importantes de um sistema computacional *Linux*. São alvos de um invasor em geral todos os diretórios de um sistema.

Ao efetuar uma intrusão, o invasor geralmente grava seus arquivos em diretórios não usuais do sistema computacional e em alguns já conhecidos por todo administrador.

Diretórios que contém bibliotecas, os programas de sistema, os arquivos de inicialização e as configurações do sistema computacional, são os mais visados (PEIXOTO, 2002).

Os diretórios que contém o nome `tmp`, também são muito visados. Sua característica padrão de conter dados sendo gravados a todo instante, pode tornar mais difícil a detecção da gravação de um diretório suspeito.

Em um sistema *Linux*, devido a sua organização baseada no *Unix*, estes diretórios são facilmente localizados por possuirem os nomes facilmente ligados a sua função, como por exemplo, `bin` que contém os programas ( binários ) e `lib` que contém as bibliotecas de sistema ( Apêndice B ).

O programa quick_lstat 2.5 foi configurado de modo a checar os diretórios importantes em ataques relacionados a intrusões mais conhecidas na literatura de segurança computacional ( Figura 3.3) (PEIXOTO, 2002).

Nos testes, o quick_lstat não conseguiu captar os dados necessários, pois apresentou um problema em um de seus módulos. O programa quick_lstat possui um módulo que efetua o teste de interfaces de rede em modo promíscuo.

Ao se instalar o *Rootkit Adore-ng*(STEALTH, 2004b) e efetuar a execução do quick_lstat o programa parava sua execução no passo referente as interfaces de rede, e era necessário matar o processo do mesmo.

Como o programa era parado em meio a execução dos testes, ele não apresentou *logs* que poderiam ser utilizados para a confecção do presente trabalho.

Assim, a opção utilizada para os trabalho, foi o software /textitAIDE. Este software é um sistema que checa a integridade de arquivos e diretórios definidos em um arquivo de configuração, onde são inseridas as políticas de atuação do mesmo.

O funcionamento do mesmo consiste na criação de um banco de dados inicial que contém as informações com o estado do sistema no momento em que se executa o *AIDE* pela primeira vez.

A partir de sua segunda execução o AIDE vai comparar o estado do sistema com o banco de dados, gerando um *log* com as inconsistências encontradas.

A instalação do *AIDE* nos ambientes de teste foi feita utilizandos os pacotes contidos no site Linuxpackages[4].

A grande vantagem da utilização deste pacote é que o mesmo utiliza a biblioteca `mhash` nativa do *Slackware 10.2* que é compilada dinamicamente. No caso da necessidade de compilar o *AIDE* através dos fontes, é necessário compilar a biblioteca `mhash`a partir dos fontes gerando uma biblioteca estática ( Figura 3.4 ).

[4]http://www.linuxpackages.net

```
# Arquivo de configuracao do lstat - Felipe Cerqueira (c)
#
#
# Sintaxe: <OPT> <PATH>
#
# O OPT varia entre 1 e 2, sendo 1 quando o PATH for um arquivo e
#                                2 quando o PATH for uma Pasta.
#
#
# Detalhes do arquivo:
# - Entre o <OPT> e o <PATH> so pode existir um "1" espaco, caso contrario, o
#   programa vai processar informacoes incorretas.
# - Observe que no final dos PATHs para Pastas e necessario a "/".
#
#        Sugestoes? fcerqueira@bufferoverflow.org
#

2 /etc/
2 /usr/sbin/
2 /usr/bin/
2 /bin/
2 /sbin/
2 /proc/
2 /lib/
2 /usr/lib/
2 /usr/src/
2 /dev/

# Put your password here
# Example: $ klFeLEuNcVr4g
# Use make_password to obtain your own password
#

$ devZtlCdGTjds

#
# EOF
```

**Figura 3.3:** Arquivo de configuração do quick_lstat

Após a instalação é necessário configurar o software *AIDE*. O *AIDE* possui um arquivo de configuração onde a sintaxe é bem simples. Na Figura 3.5 pode-se observar uma enorme gama de opções que as regras do *AIDE* podem possuir.

```
# tar -xvzf mhash-0.9.4.a.tar.gz
# cd mhash-0.9.4
# ./configure -enable-static
# su -
# make
# make install
# tar -xvzf aide-0.11-rc2.tar.gz
# cd aide-0.11-rc2
# ./configure \
  --sysconfdir=/etc/aide \
  --with-config_file=\
  /etc/aide/aide.conf \
 --with-mhash
# su -
# make
# make install
```

**Figura 3.4:** Processo de instalação do AIDE

```
p -> permissão de arquivo
i  -> inode
n -> quantidade de vínculos
u -> propriedade do usuário
g -> propriedade do grupo
s -> tamanho do arquivo
m-> última alteração
a -> último acesso
c -> alteração do inode
S -> alteração do tamanho
```

**Figura 3.5:** Descrição das regras do AIDE

O *AIDE* ainda fornece uma opção de criar *aliases*, que podem ser ententidos como um grupo de regras. Na Figura 3.6, é possível visualizar uma série de exemplos de como concatenar regras para uso no arquivo de configuração.

```
E -> tudo é ignorado
L -> p+i+n+u+g
R -> p+i+n+u+g+m+c+md5
> -> p+i+n+u+g+S
```

**Figura 3.6:** Concatenação de regras

O *AIDE* utiliza vários algoritmos de hash : `md5, sha1, md1600, tiger, crc32, hava1 e gost`. Os mais utilizados são os algoritmos `md5` e `sha1`.

O pacote já possui um arquivo de configuração pré-configurado, que pode foi utilizado como base para o arquivo utilizado nos ambientes de teste ( Apêndice C ). No arquivo foi criada uma regra chamada `Regra` e em vários diretórios.

O sinal de `!` quando colocado em frente a diretórios ou arquivos, funciona como uma negação, ou seja, eles não serão testados durante o processo de funcionamento do /textitAIDE.

Após a configuração do *AIDE* é necessário criar o banco de dados. Através do comando `aide -i`, o *software* gera um banco de dados com o estado atual do sistema. Este banco de dados vai ser utilizado todas as vezes que o *software* for executado, gerando um log com as inconsistências encontradas.

Por padrão o *AIDE* cria um arquivo no diretório `/var/lib/aide.db.new`. Este arquivo deve ser renomeado para `aide.db` para que o programa consiga utilizá-lo durante o seu funcionamento.

Foi adicionado no *cron* do sistema computacional o comando do software *AIDE* utilizado para gerar os *logs* ( Figura 3.7. Esta utilização do cron se faz necessária pois o software *[*AIDE não possui um *daemon* específico que faz a varredura do sistema em intervalos pré-definidos.

Assim, a utilização do *cron* é importante para se ter uma leitura do sistema de arquivos em intervalos de tempos pré-definidos.

```
00 23 * * * /usr/bin/aide -C > /var/log/aide.conf
```

**Figura 3.7:** Regra adicionada no cron referente ao AIDE

### 3.4.2 Snoopy Logger

Em uma análise utilizando-se da filtragem da *Sytem Call* `execve()`, é importante lembrar que os invasores utilizam-se da modificação desta chamada , através do método que consiste na substituição da função `execve()` do *kernel*.

Como o modelo de syscalls do kernel é independente de plataforma, ele contém um redirecionador interno da função `sys_execve()` para a `do_execve()`.

Quando a função `do_execve()` é requisitada pelo *kernel*, ela chama uma função chamada `open_execve()`. Esta função é que é modificada pelo invasor. Ela é modificada de modo que ela chame um programa diferente do que originalmente foi requisitado pelo usuário.

O trunfo, é que os *ELFs* dos códigos maliciosos, sobre-escrevem os ELFs do programa original, tornando mais difícil a detecção de um redirecionamento de execução.

Um *ELF* é um formato de binário ( binário , que é a base dos executáveis mais comuns do Linux.

Um método para se detectar esta subversão de funções é o uso do *Snoopy Logger*[5]. Embora seja extremamente simples, o *Snoopy Logger* é capaz de gerar *logs* de todos os comandos executados por usuários do sistema ou apenas os comandos executados pelo usuário *root*.

O programa funciona como um agente, que loga todas as chamadas de executáveis, no `/var/auth/log`. Isto fornece ao perito, em uma *análise post mortem*, a possibilidade de saber quais executáveis foram chamados durante um certo período e assim, conseguir dados de subversão de funções do *kernel*, através do carregamento de *LKMs*, via comando `modprobe`, por exemplo.

O método utilizado pelo *Snoopy Logger*, fornece ao perito dados que ele possa correlacionar, com outros que ele pode conseguir, através da análise dos *ELFs* abertos por cada processo, através dos seus ponteiros.

Esta consulta pode ser comparada a feita utilizando o comando `grep` no arquivo `System.map`, que, em geral, está localizado no diretório `/boot` das distribuições *Linux* ( Figura 3.8 ).

```
bash-3.00$ grep formats /boot/System.map
c039050c b formats
c03906c0 b quota_formats
c0391bc0 B cvf_formats
```

**Figura 3.8:** Coletando ponteiros de ELF no arquivo System.map

A instalação do *Snoopy Logger* na distribuição *Slackwar 10.2* é feita através da compilação do código fonte diretamente no sistema operacional aonde o mesmo vai detectar os redirecionamentos de execução.

## 3.5 Ferramentas Pós-Intrusão

Mesmo com um sistema concebido previamente para fornecer *informações* ao perito ou ao administrador, é necessário ter em mãos um grupo de ferramentas que possam ajudar a obter informações em um sistema invadido.

[5] http://sourceforge.net/projects/snoopylogger/

As ferramentas apresentadas nesta seção, podem ser utilizadas em perícias *Post Mortem* ou em perícia nos sistemas ainda ligados ( *Live Analysis*.

### 3.5.1 Chkrootkit

O programa *chkrootkit* é uma ferramenta interessante para uma *análise forense* relacionada relacionada com *rootkits*. O *chkrootkit* consiste de um conjunto de ferramentas que tem como função testar a presença de *rootkit* instalados em um sistema computacional, usando a técnica de comparação com assinaturas.

Estas assinaturas estão implementadas em cinco programas : `chkrootkit`, `chklaslog`, `chkwtmp`, `ifpromisc`, `chkproc`.

- O Chkrootkit : é o primeiro programa chamado. Consiste de um shell script que contém chamadas para todos os outros programas do pacote. Este shell script implemente, dentro de seu codigo, testes que procuram por características, dentro de todo o sistema operacional, que possam caracterizar a presença de algum *LKM malicioso, rootkits ou worms*.

- Chklastlog : é uma ferramenta que verifica se houve alguma mudança no arquivo lastlog do sistema. O `/var/log/lastlog` guarda todas as entradas de usuários que logaram recentemente no sistema. O programa percorre todas as entradas do arquivo `/var/log/wtmp` e as compara com cada nome de usuário encontrado no `/var/log/lastlog`. Caso haja alguma discrepância, é emitido um relatório com o número de inconsistências encontradas.

- Chkwtmp : ferramenta que verfica indícios de remoções de entrada no arquivo `/var/log/wtmp`, retornando em seu relatório o número de inconsistências encontradas. O teste se baseia em um método utilizado por várias ferramentas que ao apagar alguma entrada no `/var/log/wtmp`, preenchem esta entrada com zero. A ferramenta, testa quantas linhas contém zero, e retorna o aviso ao final.

- Ifpromisc : Este programa testa se há interfaces de rede em modo promíscuo, estado geralmente associado a instalação de *sniffers* em máquinas comprometidas. Obtém do kernel a listas de interfaces de rede da máquina e para cada interface, testa suas flags para a condição IF_PROMISC.

- chkproc : Alguns sistemas Unix implementam o process filesystem, que está geralmente no diretório `/proc`. Este sistema de arquivos contém as informações referentes a cada processo ativo no sistema. A chamada `readdir` é usada para obter-se o conteúdo de um diretório.

Alguns LKMs modificam esta chamada para que ocultem diretórios referentes a Rootkits.
Embora ocultos para `readdir`, o diretório pode ser lido com a chamada `chdir`.
O programa chkproc, tenta, sequencialmente, um `chdir` em cada um dos diretórios que representa um dos processos de sistema ( de 1 a 653335 ), e toda vez que o `chdir` é bem sucedido, o PID relacionado ao processo é colocado em uma lista. Estes PID são relacionados com a saída do comando `ps` e as chamadas `readdir`. Se houver alguma discrepância, é gerado um log.

### 3.5.2 De-termine

De-termine é uma ferramenta desenvolvida pelo próprio criador do *rootkit Adore-ng*.

Ela ajuda os administradores a procurar processos escondidos pelo *rootkit*, baseando-se nos conceitos que o mesmo utilizou ao criar o *Adore-ng*.

### 3.5.3 rkscan

Raul Silez ( (PELáEZ, 2004) ), cita um programa que podem ser utilizado para obter informações referentes ao *Rootkit Adore-ng*, em um sistema computacional comprometido. O nome deste programa é `rkscan`.

O programa `rkscan` é um pequeno *scanner* para *rootkits*. Ele detecta um pequeno conjunto de *rootkits* que são o *Adore* em suas versões 0.14, 0.2b e 0.24 e um outro *rootkit* de nome *knark* ( (MILLER, 2001) ).

Ele tem seu foco especializado em algumas características. Em relação ao *Adore* ele usa a *system call* do *kernel* `setuid()` para checar se o *rootkit* está em funcionamento, usando um parâmetro específico que possui o valor “31337+2” na versão 0.14 e “61855” na versão 0.24.

Como estes valores são modificados em tempo de compilação ( e também podem ser modificados em tempo de execução ), o *scanner* utiliza metodos de força-bruta para testar todas as possíveis variações procurando uma mensagem que possa caracterizar a existência do *rootkit* dentro do sistema computacional.

A importância deste ferramenta em um kit de resposta, é pela sua especialização na detecção de versões específicas do *rootkit Adore-ng*.

### 3.5.4 Distribuição FIRE

A distribuição *FIRE*, anteriormente conhecida como *Biatchux*, é um *Live CD* que contém 196 ferramentas voltadas para a análise forense e *Pen Test*[6].

O *FIRE* possui seis tipos de ferramentas básicas :

- Ferrramentas básicas do sistema operacional.
- Ferramentas de forense e recuperação de dados.
- Resposta a incidentes.
- *Pen Test.*
- Associação binária estática.
- Escaneamento de vírus.

É uma distribuição baseada no Knoppix[7], utilizada atualmente para criar diversos live cds.

A grande vantagem do *FIRE* é limitar o tempo que o perito teria que dedicar a procura de ferramentas na internet para criar seu próprio *Toolkit*. O *FIRE* possui dois *Tooltkits* de análises forense muito famosos em sua estrutura, o *TCT*, *The Coroner's Toolkit*[8] e o *The Sleuth Kit*[9].

Ambos fornecem uma gama de ferramentas grande para análise de sistemas de arquivos, que podem ser extremamente úteis ao perito durante uma análise.

A ferramenta das análises, a distribuição *FIRE*, permite acesso a suas ferramentas via *X Server*, em uma interface muito bem construída. Nela, é possível através de um clique com o botão direito do mouse, acessar um menu, onde amigavelmente, o perito pode realizar suas atividades.

Este método, de acesso local, via *X server*, é indicado para Perícias onde não é possível o acesso remoto a máquina. Um exemplo seria uma descoberta de uma invasão e o perito ser requisitado no local o mais rapidamente possível.

A outra opção que o *FIRE* permite é o acesso a todas suas ferramentas via ssh. Ao contrário da primeira opção, o acesso a estas ferramentas ainda não está consolidado.

A documentação não é clara sobre os processos de acesso aos programas, e isto, deixa ao perito a responsabilidade de descobrir por si só, onde estão guardados os scripts que permitem acesso as ferramentas forenses do *FIRE*.

[6] Penetration Test - testes de penetração em sistemas computacionais.
[7] http://www.knoppix.org
[8] http://www.porcupine.org/forensics/tct.html
[9] http://www.sleuthkit.org

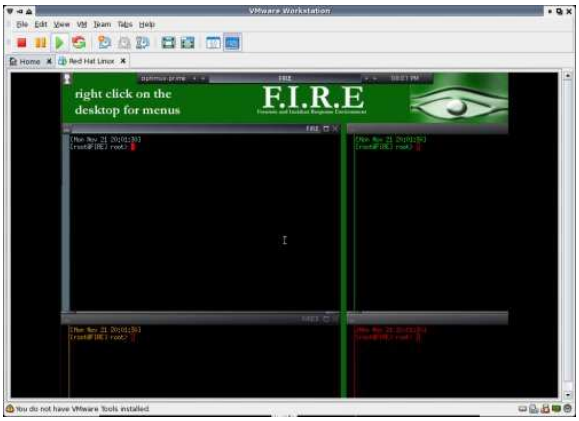


**Figura 3.9:** Menu gráfico da distribuição FIRE

No FAQ[10] do site oficial, só encontramos uma referência a um script chamado startmenu.

O *startmenu*, caso o perito não tenha efetuado nenhuma operação via X, deve ser chamado para configuração de serviços básicos do sistema operacional.

O grande problema desta interface, é, que, ela não fornece nenhum acesso ao principal do *FIRE*. As ferramentas de perícia *FORENSE*.

Uma busca cuidadosa no diretório onde está localizado o programa *startmenu*, leva a uma série de diretórios. Nestes diretórios, é possível encontrar uma série de scripts ( todos usando interface *DIALOG*[11], o que os torna mais amigáveis ) que fornece acesso as ferramentas do sistema.

Neste diretório é possível visualizar uma série de sub-diretórios, que pelo nome já mostram ao perito qual a sua função. Um exemplo é o diretório forensics, onde está a interface de acesso a ferramentas forenses.

Um grande problema, neste método de acesso ao *FIRE*, via console remoto ( *ssh* ), é que não há um script que centraliza o acesso aos diversos tipos de ferramenta que ele contém. Não é possível usar ferramentas forenses, e depois, via interface *dialog*, seguir para as ferramentas de *pen test*.

Para isto, o perito teria que fechar a interface de ferramentas forenses e ir para o diretório onde estão as ferramentas de *Pen Test*, e assim, usá-las.

O *FIRE* ainda não é uma distribuição que teve seu desenvolvimento terminado. Sua versão ainda está distante de se tornar uma versão 1.0 ( a versão usada nos testes foi a 0.3.5 ), o que caracteriza que sua maturidade ainda não foi alcançada.

[10] http://fire.dmzs.com/?section=faq

[11] http://freshmeat.net/projects/dialog/

Por este motivo, todos os problemas e eventuais bugs encontrados durante o teste, foram relevados, pois podem estar corrigidos em próximas versões.

Nos testes com o *rootkit Adore-ng* a distribuição *FIRE* foi utilizada principalmente por conter em sua listas de ferramentas do *Software Authopsy*, altamente indicado para levantamente de *timelines* dentro de um sistemas computacional comprometido.

### 3.5.5 Autopsy

O *Autopsy*, é uma ferramenta de auditoria forense, com interface web. Tem suporte aos sistemas de arquivos *NTFS, FAT, UFS1/2, Ext2/3*. Faz parte do *Sleuth Kit* e pode ser utilizado tanto para *live analysis* como para *post mortem analysis*.

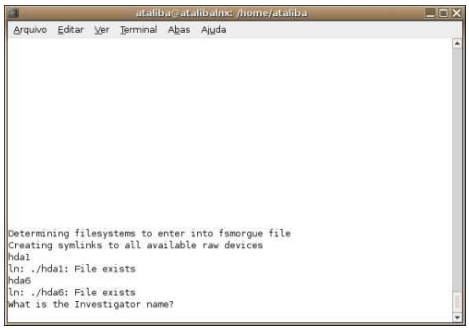


**Figura 3.10:** Tela de configuração do Autopsy

A procura de evidências pode ser feita de acordo com as seguintes técnicas :

- Listagem de arquivos : possibilidade de analisar arquivos e diretórios, inclusive aqueles apagados.
- Conteúdo do arquivo : o conteúdo de um arquivo pode ser visto em hexadecimal, raw, ou em strings ASCII. Quando é possível se entender o conteúdo do arquivo, o Autopsy limpa o mesmo, para evitar maiores problemas ao sistema.
- Hash Databases : mantém uma database com os hashes de toda os arquivos e sistemas de arquivos.
- Organização por tipos de arquivos : organiza os arquivos de acordo com as assinaturas de tipos de arquivos que o programa possui. O Autopsy só descompacta as imagens.

- Timeline das ações : capaz de criar um timeline, do mesmo modo que pode ser criado em console usando as ferramentas mac-robber e mactime.
- Procura por palavras : procura uma string específica dentro do sistema de arquivos.
- Análise de Meta Data : as estrutura de Meta Data contém detalhes sobre arquivos e diretórios. O Autopsy é capaz de ler esta estrutura e mostrar em um relatório. Utilizado na recuperação de arquivos deletados.
- Data Unit Analisys : os data units nada mais são que os arquivos guardados no hard disk. O Autopsy permite que se veja cada arquivo e o que está alocado nele.
- Detalhes do sistema de arquivos : os detalhes do sistema de arquivos podem ser vistos, incluindo o lay out do disco e o time activity, ou seja, a linha do tempo das ações efetuadas no sistema de arquivos.

No Autopsy as investigações são organizadas como cases, onde cada cada um deles pode possuir um ou mais hosts.

Uma sequência de eventos pode ser organizada de acordo com um IDS ou até logs de firewall. Os relatórios da ferramenta são gerados em formato ASCII, facilmente lidos em qualquer sistema operacional.

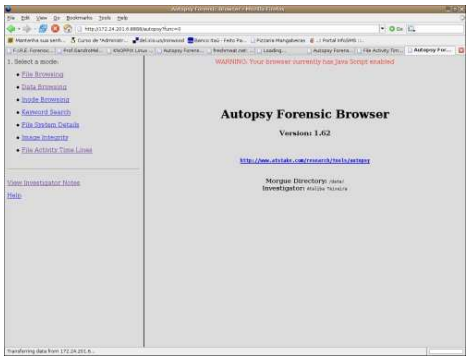


**Figura 3.11:** Tela inicial do Autopsy

Na tela inicial, ao clicar na primeira opção, o recurso de *File Browsing* permite navegar no conteúdo de um sistema de arquivos. Neste sistema de arquivos é possível visualizar cada arquivo que esteja no mesmo ou deletado.

Nas análises efetuadas, foi possível encontrar arquivos deletados, podendo em alguns casos, inclusive, visualizar seu conteúdo. Isto pode acontecer, pois como

já citado no capítulo 2 e 3, o arquivo só é realmente apagado de um disco rígido, quando outro arquivo sobre-escreve a área em que o mesmo se encontra guardado.

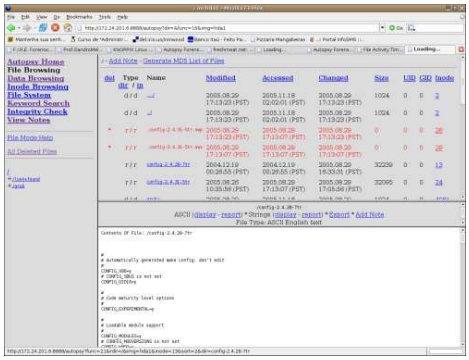

**Figura 3.12:** Autopsy listando um arquivo e seu conteúdo

O Autopsy fornece outras opções importantes em relaçã ao arquivo. Ele fornece a possibilidade de navegar por cada i-nodo do disco, na sua opção Inode Browsing. Isto pode ser importante, quando o perito já tem em mãos em qual i-nodo estava o arquivo, e somente quer visualizá-lo.

O número de blocos de cada arquivo é fornecido também, nos relatórios sobre cada arquivo em disco.

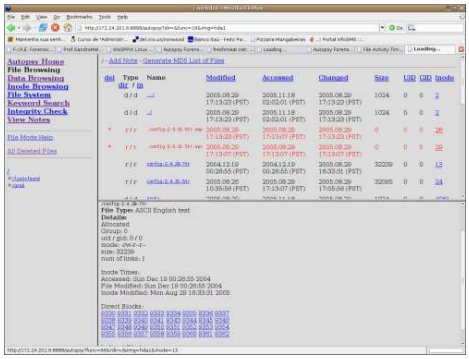

**Figura 3.13:** Autopsy listando os blocos ocupados por um arquivo

A opção File System é utilizada para gerar um relatório completo do sistema de arquivos.

O Autopsy mostrou-se uma opção interessante para que o perito possa comparar com os dados obtidos com as ferramentas console e logs de outras ferramentas.

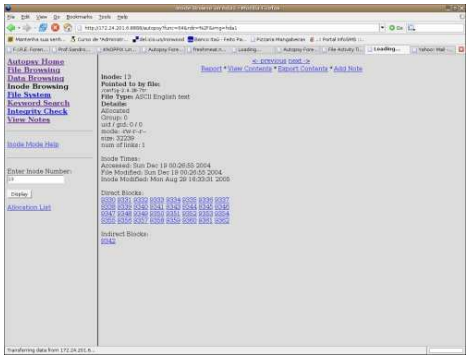

**Figura 3.14:** Listando o conteúdo de um inodo

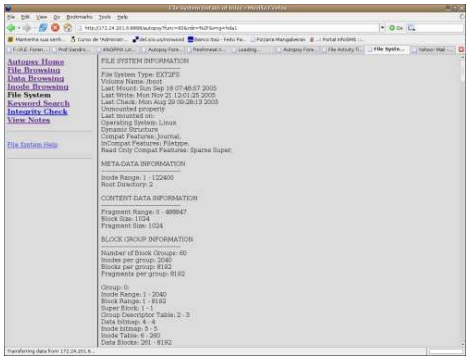

**Figura 3.15:** Relatório do sistema de arquivos

Em seu conteúdo, o programa fornece também uma opção de construir uma timeline das ações. Assim, tendo as duas timelines o perito tem condições de efetuar comparações e chegar a um resultado positivo em sua análise.

## 3.6 Proposta de um ambiente de Teste

Para coletar as informações referentes ao comportamento dos *rootkits* em sistemas operacionais *Linux*, foi necessário a criação de um pequeno laboratório.

Este laboratório foi criado a partir de máquinas virtuais, utilizando o software *VMWARE*[12]. Estas máquinas virtuais foram instaladas utilizando a distribuição

[12]http://www.vmware.com

Linux *SLACKWARE*[13], onde foram inseridos os softwares *AIDE* 3.4.1 e *Snoopy Logger* 3.4.2 (ATíLIO, 2003).

Em termos gerais, uma máquina virtual é um *software* que cria um ambiente entre a plataforma e o usuário, onde é possível criar um novo ambiente computacional. Este ambiente computacional é um ambiente fictício, onde todos os componentes de hardware são *simulados*.

Um exemplo de máquina virtual, fartamente encontrado na internet são os *emuladores* de *vídeogames* antigos. O processo de simulação é transcrever as instruções de um processador alvo, para o processador no qual ele está sendo executado (SILBERSCHATZ; GALVIN, 2000).

De acordo com Silberchatz (SILBERSCHATZ; GALVIN, 2000) existem duas principais vantagens no uso de máquinas virtuais. Primeiro, para proteger os recursos do sistema, criando um nível robusto de segurança. Segundo, as máquinas virtuais permitem que o desenvolvimento de atividades de desenvolvimento e testes possam ser feitos sem atrapalhar as operações normais do sistema (GEUS, 2004).

Outra vantagem no uso de máquinas virtuais é a introdução de maiores níveis de segurança baseados no isolamento do usuário administrador da máquina virtual, pela adição de mais uma camada de proteção ao acesso a máquina real e por sua fácil restauração em caso de problemas (GEUS, 2004).

A máquina virtual cria uma dupla camada de isolamento entre as aplicações que estarão sendo executadas no sistema operacional convidado e o sistema operacional hospedeiro. Assim, um atacante ao obter sucesso em subverter uma aplicação qualquer e dominar a máquina virtual, precisaria subverter também o *software* que está executando aquela máquina virtual, para finalmente, chegar a máquina real (GEUS, 2004).

O *software VMWARE* permite a criação de clones das máquinas virtuais criadas. Assim, em experimentos que envolvam a subversão do sistema operacional, como o presente trabalho, é possível recuperar o sistema em seu estado inicial, para que seja possível testar as diversas interações de um *malware* com o sistema operacional hospedeiro.

Para a criação da máquina virtual, baseada no *rootkit Adore-ng*, foi necessário efetuar o *download* de uma versão operacional ( versão 0.53 ) do mesmo, que foi encontrada em um dos sites do autor [14].

A Figura 3.16, mostra o diretório criado após a descompactação do *rootkit*. A partir deste diretório, para a criação dos binários para o *kernel* da máquina, foi utilizado o comando `configure` que é um script que configura as opções necessárias para o funcionamento do *rootkit Adore-ng* ( Figura 3.17 ).

[13] http://www.slackware.com

[14] http://stealth.openwall.net/rootkits/

```
root@atalibateste:/usr/src/. /adore-ng# ls
CVS/                                    adore-ng-2.6.c  libinvisible.c
Changelog                               adore-ng.c      libinvisible.h
FEATURES                                adore-ng.h      relink*
LICENSE                                 adore-ng.mod.c  relink26*
Makefile                                adore-ng.o      startadore*
Makefile.2.6                            ava*            symsed*
Makefile.2.6.gen                        ava.c           symsed.c
Makefile.gen                            cleaner.c       visible-start.c
Makefile_Tue_Jan_10_19:48:19_BRST_2006  cleaner.o       zero.o
README                                  configure*
README.26                               irq_vectors.h
root@atalibateste:/usr/src/. /adore-ng#
```

**Figura 3.16:** Diretório com os fontes do Adore-ng

Após a configuração, é necessário utilizar o comando `make`, para gerar os binários necessários ao funcionamento do *rootkit*. O *rootkit Adore-ng* é iniciado utilizando o comando `startadore`, contido no diretório do com os fontes. Este aplicativo, carrega os módulos necessários ao funcionamento do *rootkit*, para que o mesmo possa efetuar suas funções de manipulação do *VFS* do *Linux* ( Figura 3.19 ).

Uma das funcionalidades mais interessantes do *rootkit Adore-ng* após seu carregamento, é, que, o mesmo torna o diretório onde seus binários estão localizado, invisível. Um comando `ls` executado no local onde estão os binários deste programa, simplesmente, não irá mostrar mais este diretório, tornando a perícia um pouco mais dificultada ( Figura 3.18 ), já que tal artefato trabalha diretamente nas funções de *kernel* do sistema computacional.

Uma das vantagens para o invasor, é que o *rootkit Adore-ng* fornece a possibilidade de esconder processos, inclusive manipulando o diretório `/proc`, que é o local do sistema *Linux* onde estão localizadas as informações sobre os processos.

Assim, foi utilizado um *backdoor* ( Apêndice D ) de funcionamento básico e que atendeu bem aos resultados que eram esperados. Na Figura 3.20 o aplicativo `ava` foi utilizado para tornar o processo do *backdoor* invisível.

Em uma invasão, o administrador teria problemas para encontrar anomalias em seu sistema computacional, pois, ao utilizar o comando `ps`, ele não encontraria nada diferente, já que o processo está invisível.

Outro ponto importante em uma invasão é esconder o diretório onde estão os artefatos utilizados para comprometimento de um sistema. Assim, o mesmo aplicativo `ava` pode ser utilizado para este fim. A opção utilizada em conjunto com

```
Using byte-width of 4 for UID/GID

Starting adore configuration ...

Checking 4 ELITE_UID + ELITE_GID ... found 1742629802, 599923811
Checking 4 SMP ... NO
Checking 4 MODVERSIONS ... NO
Checking for kgcc ... found cc
Checking 4 insmod ... found /sbin/insmod -- OK

Loaded modules:
pcmcia_core            39172   0
ide-scsi                9392   0
pcnet32                17028   1
mii                     2272   0 [pcnet32]
crc32                   2880   0 [pcnet32]
agpgart                45508   0 (unused)


Since version 0.33 Adore requires 'authentication' for
its services. You will be prompted for a password now and this
password will be compiled into 'adore' and 'ava' so no further actions
by you are required.
This procedure will save adore from scanners.
Try to choose a unique name that is won't clash with filenames in /proc.
Password (echoed):
Preparing '.' for hiding ...

Creating Makefile ...

#*** Edit adore-ng.h for the hidden services ***
```

**Figura 3.17:** Tela de configuração do Adore-ng

```
root@atalibateste:/usr/src/. # ls
adore-ng-0.53.tgz  backdoor*  backdoor.c
root@atalibateste:/usr/src/. # pwd
/usr/src/.
root@atalibateste:/usr/src/. #
```

**Figura 3.18:** Diretório após o carregamento do rootkit Adore-ng

```
root@atalibateste:/usr/src/. /adore-ng# ./startadore
root@atalibateste:/usr/src/. /adore-ng# ./ava
Usage: ./ava {h,u,r,R,i,v,U} [file or PID]

       I print info (secret UID etc)
       h hide file
       u unhide file
       r execute as root
       R remove PID forever
       U uninstall adore
       i make PID invisible
       v make PID visible

root@atalibateste:/usr/src/. /adore-ng#
```

**Figura 3.19:** Iniciando o rootkit Adore-ng

```
root@atalibateste:/usr/src/. /adore-ng# ./ava
Usage: ./ava {h,u,r,R,i,v,U} [file or PID]

       I print info (secret UID etc)
       h hide file
       u unhide file
       r execute as root
       R remove PID forever
       U uninstall adore
       i make PID invisible
       v make PID visible

root@atalibateste:/usr/src/. /adore-ng# ps ax | grep kswap | grep -v grep
    4 ?        S      0:00 [kswapd]
  339 tty1     S      0:00 [kswapd] r
root@atalibateste:/usr/src/. /adore-ng# ./ava i 339
Checking for adore  0.12 or higher ...
Adore 1.53 installed. Good luck.
Made PID 339 invisible.
root@atalibateste:/usr/src/. /adore-ng# ps ax | grep kswap | grep -v grep
    4 ?        S      0:00 [kswapd]
root@atalibateste:/usr/src/. /adore-ng# ls /proc | grep 339
```

**Figura 3.20:** Aplicativo ava sendo utilizado para tornar um processo invisível

este aplicativo é a h, que torna um arquivo ou diretório invisíveis para o sistema ( Figura 3.21 ).

```
root@atalibateste:/usr/src/. /adore-ng# ./ava h "/usr/src/. "
Checking for adore  0.12 or higher ...
Adore 1.53 installed. Good luck.
File '/usr/src/. ' is now hidden.
root@atalibateste:/usr/src# ls -la
total 20
drwxr-xr-x   6 root root 4096 2006-01-10 19:47 ./
drwxr-xr-x  18 root root 4096 2005-09-16 07:26 ../
lrwxrwxrwx   1 root root   12 2006-01-10 18:35 linux -> linux-2.4.31/
drwxr-xr-x  15 root root 4096 2005-06-05 22:17 linux-2.4.31/
drwxr-xr-x   7 root root 4096 2003-10-29 04:08 rpm/
drwxr-xr-x   2 root root 4096 2005-06-05 22:25 speakup-2.4.31/
root@atalibateste:/usr/src#
```

**Figura 3.21:** Escondendo o diretório que contém os artefatos utilizados para a invasão

A partir deste ponto, os rastros de uma intrusão efetuada com o *rootkit Adore-ng* está terminada. Um invasor teria aqui um sistema apto a receber sua visita, sempre que necessário, pois haveria sempre uma porta aberta, para uma conexão sem problemas.

A partir destes passos foi criada uma máquina virtual para os testes relacionados ao *rootkit Adore-ng*.

## 3.7 Taxonomias e Resultados Obtidos

### 3.7.1 Resultados Obtidos

Uma perícia sempre é iniciada a partir de observações de funcionamentos anômalos em um sistema computacional. Tais comportamentos podem ser utilização excessiva da banda a partir de um *host* na rede ou até, um aviso externo, como alguns efetuados pelo *CERT*.

O *CERT* é o Computer Emergency Response Team. Organismo criado em 1988 pela Darpa, visando tratar questões de segurança em redes, em particular na Internet.

A partir do momento em que o perito é acionado, ele irá ao local, pronto a receber uma máquina com diversos problemas e/ou, ainda com o invasor efetuando alguma ação dentro da mesma.

No caso dos *rootkits* a análise preliminar é que irá fornecer maiores informações sobre as ações efetuadas em um sistema computacional.

A ferramenta utilizada para alguns dos testes foi a distribuição *FIRE*.

Em seu cd, o *FIRE* fornece a ferramenta *chkrootkit*, que foi utilizado para obter alguma entrada que pudesse demonstrar a presença de algum *rootkit* dentro do sistema operacional. O resultado obtido ( Apêndice E não demonstrou a presença de nenhum programa malicioso dentro do sistema computacional.

Isto se deve, principalmente, pelo *chkrootkit* não possuir em sua listagem as assinaturas de mudanças efetuadas no sistema computacional pelo *rootkit Adore-ng*. Um detalhe a ser observado, é, que, apesar de possuir uma entrada *Adore* em sua listagem, esta se refere ao *worm* de mesmo nome ( (INSTITUTE, 2001) ).

Infelizmente, o *FIRE* não fornece um conjunto de ferramentas pré-compiladas de fácil acesso ao perito, somente com a montagem do cd. Assim, foi necessário retirar o *CD* do driver e efetuar os outros testes com binários compilados e obtidos de outra máquina que continha o *Slackware*.

De posse dos *logs* obtidos pela ferramenta *AIDE* foi possível observar que um diretório de nome suspeito foi criado abaixo do diretório `/usr/src/`.

Dentro deste diretório, vários arquivos suspeitos estão gravados, e que podem caracterizar a intrusão efetuada com um *rootkit* qualquer. Os nomes dos arquivos remetem a presença do *rootkit Adore-ng*, que deve ser o o alvo de estudo do perito nos próximos passos.

Como o sistema possui o *Snoopy Logger* instalado, uma pequena busca nos *logs* pode trazer informações interessantes, a serem comparadas com as obtidas nos logs do software *AIDE*.

No Apêndice F, é possível visualizar a presença do *rootkit Adore-ng* neste sistema computacional.

De acordo com os resultados obtidos através de filtragens dos *logs* com o comando `grep` nativo do próprio *Linux*, é possível observar que realmente arquivos suspeitos foram criados abaixo do diretório `/usr/src`.

Neste diretório foram guardados códigos fonte, que poderiam ser de algum *rootkit*. Quando o perito procura o diretório através do comando `ls`, ele não encontra nada, já que o invasor se preveniu tornando este diretório invisível aos comandos do *Linux*, ação também observada nos *logs* observando o uso da opção `h` do programa `ava`.

O próximo passo foi a utilização do programa rkscan. Infelizmente, nos testes efetuados, o `rkscan` não conseguiu detectar o *Adore-ng*. Como a versão utilizada nos testes foi a 0.53, não suporta por este *scanner* ele simplesmente apresentou um resultado tal qual o *chkrootkit*, sem valor para a a perícia ( Figura 3.22 ).

A última opção para detecção do *rootkit Adore-ng* foi o software `de-termine`.

```
ataliba@atalibateste:~$ ls
rkscan1.0.c
ataliba@atalibateste:~$ gcc -o rkscan rkscan1.0.c
rkscan1.0.c: In function ‘adore_scan’:
rkscan1.0.c:46: warning: comparison between pointer and integer
ataliba@atalibateste:~$ ./rkscan
-=-      Rootkit Scanner      -=-
-=- by Stephane.Aubert@hsc.fr -=-

  Scanning for ADORE version 0.14, 0.24 and 2.0b ...
  ADORE rootkit NOT DETECTED on this system.

  Scanning for KNARK version 0.59 ...
  KNARK rootkit NOT DETECTED on this system.

Done.
ataliba@atalibateste:~$
```

**Figura 3.22:** Resultados do programa rkscan

Este *software* foi único capaz de detectar e mostrar com certeza a presença deste *rootkit* no sistema computacional. Assim, se torna uma peça importante em um *kit* de resposta, por ser uma ferramenta altamente eficaz ( Figura 3.23 ).

Um dos pontos observados nos filtros efetuados no *log* `/var/log/secure` ( Apêndice F ), foi a presença do nome *backdoor*. Sendo assim, a possibilidade de um programa estar esperando conexões dentro do sistema computacional é bem grande.

Através do comando `netstat`, também nativo do sistema operacional *Linux* foi possível visualizar um estranho programa escutando na porta 20000. Os dados do comando `netstat` não são muito exatos, mas possuem a possibilidade de ser complementados com o uso do comando `lsof`.

O comando `lsof` lista os arquivos abertos por um processo, bem como também suas conexões de rede. Assim, foi interessante utilizar este comando para tentar conseguir chegar ao programa que está utilizando a porta 20000.

Infelizmente, devido ao uso do programa `ava` em paralelo com o *rootkit Adore*, as conexões de rede foram escondidas dos comandos do *Linux*, tornando esta tarefa um pouco mais difícil para o perito ( Figura 3.24 ).

Uma intrusão com o *rootkit Adore-ng* em um ambiente de produção seria muito difícil de ser detectada com pequenas ações. A presença do *AIDE* se mostrou importante, pois, as mudanças em um disco rígido são necessárias em algum momento da invasão (ATíLIO, 2003).

```
root@atalibateste:~# cd determine
root@atalibateste:~/determine# ls
CVS/  Makefile  README  detect.c  detect.h  main.c  signatures/
root@atalibateste:~/determine# make
cc -Wall -O2 -c main.c
cc -Wall -O2 -c detect.c
cc main.o detect.o -o determine
root@atalibateste:~/determine# ./determine

deter-mine LKM rootkit detector. (C) 2004 Stealth

Trying to detect hidden processes ...
Process with PID 339 does not have a appropriate /proc entry. Hidden?
Done.
Scanning /dev/mem for signatures. This may take a while ...
Signature of 'Adore/Adore-ng LKM' detected!

Unusual behaivior has been detected. Please consult the removal chapter of the README-file.

root@atalibateste:~/determine#
```

**Figura 3.23:** Resultados do programa determine

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:20000           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:37              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:113             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0:512             0.0.0.0:*
udp        0      0 0.0.0.0:37              0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
unix  3      [ ]         DGRAM                    232    /dev/log
unix  2      [ ACC ]     STREAM     LISTENING     706    /dev/gpmctl
unix  2      [ ]         DGRAM                    243

root@atalibateste:/usr/src/. # lsof | grep 20000
```

**Figura 3.24:** Resultados do comando netstat

### 3.7.2 Taxonomias Observadas

Na arquitetura em que o trabalho foi construido, não foi possível levantar muito do que um ataque com *rootkits* pode causar a um sistema computacional.

Isto aconteceu devido aos testes efetuados serem em sistemas computacionais criados especificamente para este estudo, não demonstrando realmente como um invasor poderia efetuar isto em um sistema real.

Apesar disto, as taxonomias aplicadas no projeto, são as mais divulgadas em trabalhos pela internet, como citado por Antônio Marcelo, um dos responsáveis pelo projeto Honeypot-BR **??** , em seu artigo sobre o projeto Pliscka **??**.

Um dos pontos interessantes a ser observar, é, que, com uma estrutura que mantenha um bom histórico do que é efetuado em uma rede durante períodos, é possível levantar com muito mais facilidade as características dos ataques em um sistema computacional.

Em relação ao *Adore-ng*, é importante salientar que o mesmo trabalha em espaço de *kernel*, efetuando suas mudanças de modo tão transparente, que até administradores muito experientes podem não detectá-lo com as auditorias de rotina.

Um dos modos mais interessantes para a detecção de taxonomias de ataque em sistemas computacionais é o uso de uma ferramenta como o *software Autopsy*. Ele, por suas características, pode fornecer a possibilidade de criar uma linha de tempo dos acontecimentos em um sistema computacional comprometido.

Um ataque padrão normalmente acontece com a detecção de uma vulnerabilidade em um sistema computacional e sua respectiva exploração através de um *exploit* ou outro programa específico.

Nesta fase, o invasor ainda estuda o alvo, e irá utilizar-se de scanners para obter informações relevantes sobre o alvo. Neste ponto, um dos scanners mais utilizados é o `nmap`. O `nmap` se mostra interessante nesta pré-intrusão pois o mesmo, através de sua várias técnicas de varredura, poderá fornecer ao invasor diversas informações que possam levá-lo a uma intrusão bem sucedida.

Após este levantamente de informações do alvo, o invasor irá utilizar um *exploit* para ganhar um *shell de root* no alvo. Um *exploit* é um programa que subverte *bugs* de um *software* em uma ação que o invasor queira.

Os *exploits* geralmente possuem em seus códigos, métodos utilizados em *shellcodes*. Um *shellcode* geralmente é um programa em linguagem *assembly*, que provém um *shell* ao atacante.

O método utilizado pelos *exploits* para ganhar o *shell* de *root* ( superusuário em sistemas *Linux* é o *buffer overflow*.

Os computadores possuem *chips* especiais chamados de memória. Estes *chips* tem a função de guardar informações em forma de *bits*.

Quando um programa está sendo executado os endereços, valores de variáveis e constantes ficam armazenados na memória. Para que não haja desorganização, a memória é dividida em áreas para guardar dados específicos.

Estas áreas possuem limites e são definidas como *buffer*.

A expressão *buffer overflow* significa um estouro de *buffer*. Ao se *800 ml* em um recipiente que suporte *600 ml*, ao chegar a marca de *600 ml* o recipiente transbordaria. Isto é o mesmo que acontece com uma variável em um *buffer overflow*(ANôNIMO, 2000) (NUMABOA.COM, 2004).

Ao se atribuir 30 caracteres a uma variável que só suporte 25 caracteres, temos um *buffer overflow*.

Assim, a partir da obtenção do *shell de root* a máquina se torna apta a ser controlada pelo invasor.

A partir deste momento quem trabalha é o *rootkit*. O *Adore-ng* é um *rootkit* que trabalha principalmente no espaço de kernel, ou seja, ele age no sistema computacional diretamente em seu núcleo.

Sua taxonomia principal é subverter as funções do kernel referentes ao *VFS* tornando sua detecção em uma *Live Analysis* quase impossível.

Com o *software Authopsy* foi possível, em uma análise *post mortem*, detectar os binários do *Adore-ng* e os códigos fontes referentes ao *backdoor* que se encontrava instalado no sistema computacional.

A partir deste *software* foi possível construir uma linha de tempo de como ocorreu o ataque e assim, comparando com os *logs* obtidos nos *softwares AIDE* e *Snoopy Logger* foi possível detectar que, o mais importante, em uma invasão deste tipo, é manter um sistema com métodos de detecção de intrusão eficazes.

## 3.8 Considerações Finais

A base para um sistema computacional seguro, é a intervenção humana feita de modo acertado.

Assim, o administrador deve levar em conta que servidores são máquinas que precisam ter características diferentes de uma estação de trabalho.

E o conhecimento das ações dos *rootkits* em sistemas operacionais *Linux* é uma arma para proteger-se de futuras intrusões.

# Capítulo 4

# Conclusões

Os *rootkits* são artefatos avançados utilizados para comprometer sistemas *Linux*. Assim sendo, é necessário que administradores e peritos forenses, tenham em mente que, como em cenas reais de crime, uma máquina pode guardar diversas evidências de uma intrusão.

A necessidade do conhecimento do comportamento destes artefatos é grande, pois através disto é que podem ser concebidos métodos e contra-medidas contra as ações dos mesmos.

Os *rootkits* são artefatos que ao longo dos anos estão evoluindo, e cada vez mais se faz necessário a proposta de um método para aplicação em perícias forenses relacionadas a eles.

O presente trabalho mostrou uma pequena faceta de como trabalham os *rootkits* da primeira e das atuais gerações. Como pôde ser visto, a sua detecção se torna cada vez mais difícil ao perito ou ao administrador, forçando um dos mesmos a instalar em seus sistemas computacionais diversas ferramentas para auxiliá-lo na ocorrência de uma intrusão.

O trabalho foi desenvolvido baseado em invasões pré-definidas pelo autor, tendo as perícias e estudos efetuados um caráter não prático.

Mesmo assim, é possível o o *LKM Saint Jude*, para evitar os ataques com redirecionamentos de execução.

O *Saint Jude* é um projeto que implementa um *IDS em nível de kernel*. Seu mecanismo de detecção tenta alertar sobre algum funcionamento estranho no sistema computacional, como uma tentativa de mudança de privilégio executada de um modo estranho.

É um *LKM*, que sobre-escreve algumas das funções do kernel para que possa aplicar suas regras. As suas regras, são aprendidas via funcionamento normal do

sistema, onde o *Saint Jude* vai monitorando as chamadas da função `execve()` e as vai caracterizando, associando-as a uma lista de restrição.

Quando uma regra é violada, o processo envolvido é abortado. É indicado principalmente para evitar que invasores ganhem o *shell de root*, via *buffer over-flows*.

Em trabalhos futuros, relacionados a este tema, é interessante implementar uma *honeypot* onde serão estudadas as ações de invasores em um ambiente real, e os efeitos que os *rootkits* instalados teriam nos resultados de uma perícia.

Assim, os dados obtidos seriam mais condizentes com a realidade de uma invasão no mundo real.

Além disto, é possível propor a mesma estrutura de trabalho para detecção de *rootkits* em sistemas Windows e nos *Unixes-Like* da família *BSD*.

O presente trabalho forneceu informações importantes sobre como trabalha um *rootkit* quando instalado em um sistema operacional e como deve proceder o administrador para defender melhor seus sistemas de ações deste tipo.

# Referências Bibliográficas


ANôNIMO. *Segurança Máxima*. 1. ed. São Paulo: Editora Campus, 2000.

ATíLIO, C. E. *Padrão "ACME!"para análise forense em sistemas computacionais*. Dissertação (Mestrado) — Universidade Estadual de São Paulo, 2003.

BIOFORGE. Hacking the linux kernel network stack. *Phrack Magazine*, v. 11, n. 61, January 2003.

BOVET, D. P.; CESATI, M. *Understanding the Linux Kernel*. 1. ed. Sebastopol: O'Reilly, 2003.

DEVIC, S. e. *Linux on the fly kernel patching without LKM*. 2001. Phrack.

GEUS, A. B. de Castro e Cleymone Ribeiro dos Santos e Paulo Lício de. *MV6 - Um mecanismo de transição baseado em máquinas virtuais*. 2004. WSEG2004.

IBM. *LKCD Installation and Configuration*. 2002. IBM GLOBAL SERVICES.

INSTITUTE, S. *Adore Worm*. 2001. SANS.

KLAN, O. *How to use Netfilter Hooks*. 2003. Netfilter.org.

MARCELO, A. Dormindo com o inimigo. *Revista Hacker*, v. 1, n. 10, Outubro 2000.

MAXWELL, S. *Kernel do Linux*. 1. ed. São Paulo: Makron Books, 2000.

MILLER, T. *Analysis of the KNARK Rootkit*. 2001. SecurityFocus.

MURILO, K. S.-J. e N. Métodos para detecção local de rootkits e módulos de kernel maliciosos em sistemas unix. In: *Anais do III Simpósio sobre Segurança em Informática*. São José dos Campos: SSI'2001, 2001.

NUMABOA.COM. *O que é um buffer overflow?* 2004. Aldeia NumaBoa.

PEIXOTO, J. F. de. *Honeynet : um ambiente para análise de intrusão*. Dissertação (Mestrado) — Universidade Estadual Paulista, 2002.

PELáEZ, R. S. *Linux Kernel Rootkits : protecting the system's "Ring Zero"*. 2004. GIAC.

PRAGMATIC. *(nearly)The Complete Linux Loadable Modules*. 1999. The Hackers Choice.

SCHILDT, H. *C Completo e Total*. 2. ed. São Paulo: Makron Books, 1997.

SCHWEITZER, D. *Incident Computer Forensics Toolkit*. 1. ed. United States: Wiley Publishing Inc., 2003.

SILBERSCHATZ; GALVIN. *Sistemas Operacionais - Conceitos*. 1. ed. São Paulo: Prentice Hall, 2000.

STEALTH. Kernel rootkit experiences and the future. *Phrack Magazine*, v. 11, n. 61, April 2003.

STEALTH. *The Adore-ng Rootkit*. 2004. TESO.

STEALTH. *The Adore Rootkit*. 2004. TESO.

TRUFF. Infecting loadable kernel modules. *Phrack Magazine*, v. 11, n. 61, February 2003.

UCHôA, K. C. A. *Introdução à Cibercultura*. 2004. UFLA.

VáRIOS. *Ciência da Computação*. 2005. Wikipedia.

WOODHULL, A. S. T. e A. S. *Sistemas Operacionais : Projeto e Implementação*. 2. ed. São Paulo: Bookman, 2000.

ZIVIANI, N. *Projeto de Algoritmos com implementações em Pascal e C*. 5. ed. São Paulo: Pioneira Informática, 1999.

# Apêndice A

# Exemplo de log do quick_lstat

```
LSTAT IDS SYSTEM
   Date:  Sun Jan 8 19:55:27 2006
   Database found, verifing...
   Warning :  System Changed:
   => /etc/mtab - last change:  Fri Sep 30 11:05:06 2005 -> Wed Nov
16 18:41:33 2005
   => /etc/shadow.YaST2save - size:  953 -> 1005
   => /etc/shadow.YaST2save - last change:  Sat Sep 10 00:46:00 2005
-> Fri Dec 16 17:16:00 2005
   => /etc/shadow.YaST2save - md5 digest:  36c7d0b7d76bba6bf1f296833e084424
-> 48d29505d59cd6cd833b3851ff9ae094
   => /etc/crontab - size:  344 -> 373
   => /etc/crontab - last change:  Sat Oct 15 13:59:45 2005 -> Sat
Oct 22 10:01:55 2005
   => /etc/crontab - md5 digest:  259cde8d460939e3c0eb9521f5935682
-> 61b3777ca6202a47561e49a3146316e5
   => /etc/defkeymap.map - size:  48074 -> 47988
   => /etc/defkeymap.map - last change:  Fri Sep 30 11:05:00 2005
-> Wed Nov 16 18:41:27 2005
   => /etc/defkeymap.map - md5 digest:  9b41fdf8c062ed09ae249b3baacc0305
-> fbd848b4fb345e1914223c540f9efa07
   => /etc/group - size:  676 -> 716
   => /etc/group - last change:  Tue Sep 13 12:38:39 2005 -> Fri Dec
16 17:31:55 2005
   => /etc/group - md5 digest:  cad8993a19256b707194c49b1f67c8bb ->
87c26f7edf02bca1f76b180a0ec32958
```

```
   => /etc/defkeymap.name - last change:  Fri Sep 30 11:05:00 2005
-> Wed Nov 16 18:41:27 2005
   => /etc/blkid.tab - last change:  Fri Sep 30 11:04:54 2005 -> Wed
Nov 16 18:41:20 2005
   => /etc/blkid.tab - md5 digest:  ada0a474d975d0968151923be853539e
-> 3d5363131a2594155adaaa21daf5af17
   => /etc/shadow.old - size:  1093 -> 1094
   => /etc/shadow.old - last change:  Tue Oct 11 11:00:34 2005 ->
Fri Dec 16 17:31:55 2005
   => /etc/shadow.old - md5 digest:  ccbc25ff14c136ab9e6287254de97b04
-> 346b01b46ba250aea8a7266978d1432a
   => /etc/adjtime - last change:  Fri Sep 30 11:04:49 2005 -> Fri
Nov 18 13:13:01 2005
   => /etc/adjtime - md5 digest:  b9c1a19cc7ba802c9c2a4fd280b41fb2
-> 501f8b3e1c96e99afee989f815c7f789
   => /etc/passwd - size:  1487 -> 1522
   => /etc/passwd - last change:  Tue Sep 27 20:04:51 2005 -> Fri
Dec 16 17:31:55 2005
   => /etc/passwd - md5 digest:  1a80ae63666a785a849dcec9af875bfc
-> 83eb0e2b2034201d20d71ae00db43367
   => newfile :  /etc/oratab - uid:  1009 gid:  1004 permissions:
100644 md5 digest:  7f4c061d7d9258ee236aa12f14095dfb
   => /etc/group.YaST2save - size:  618 -> 680
   => /etc/group.YaST2save - last change:  Sat Sep 10 00:48:56 2005
-> Fri Dec 16 17:16:23 2005
   => /etc/group.YaST2save - md5 digest:  0c51eeb08de7d19126a84698cb52b544
-> e64d73f1712289809c8e645d1a1ad37b
   => /etc/sysctl.conf - size:  266 -> 375
   => /etc/sysctl.conf - last change:  Mon Aug 22 12:02:56 2005 ->
Fri Dec 16 17:31:40 2005
   => /etc/sysctl.conf - md5 digest:  f50469b427e1db30d8e06612ed50ff53
-> cda685ebcfc83a011e4fd6330f5dd790
   => /etc/papersize - last change:  Mon Oct 17 08:18:37 2005 -> Fri
Dec 16 17:31:28 2005
   => /etc/shadow - size:  1093 -> 1122
   => /etc/shadow - last change:  Tue Oct 11 11:00:34 2005 -> Fri
Dec 16 17:31:55 2005
   => /etc/shadow - md5 digest:  f3ddc523caeca789cb69d2b04743dcee
-> 11a46f1b75faf5717359e3bc251728f5
```

```
   => /etc/group.old - size:  669 -> 707
   => /etc/group.old - last change:  Tue Sep 13 12:38:39 2005 -> Fri
Dec 16 17:31:55 2005
   => /etc/group.old - md5 digest:  1e7f10d28180c2272453d493fab3636a
-> e26bc6cdc030f006175528b5b243daa6
   => /etc/blkid.tab.old - last change:  Fri Sep 30 11:04:54 2005
-> Wed Nov 16 18:41:20 2005
   => /etc/blkid.tab.old - md5 digest:  756caa09734763837c1687402921b890
-> 3d5363131a2594155adaaa21daf5af17
   => /etc/ld.so.cache - size:  26052 -> 26795
   => /etc/ld.so.cache - last change:  Mon Oct 17 08:18:37 2005 ->
Fri Dec 16 17:31:56 2005
   => /etc/ld.so.cache - md5 digest:  08058aaa36509ff540524e76d2770dcb
-> 595eb0210eee6d06a80bd683df6cfb7c
   => /etc/passwd.YaST2save - size:  1329 -> 1423
   => /etc/passwd.YaST2save - last change:  Sat Sep 10 00:46:00 2005
-> Fri Dec 16 17:15:59 2005
   => /etc/passwd.YaST2save - md5 digest:  0e412e651eec65a0e34d3295536411fa
-> 63a801da37929588fc97cd90e3e21b03
   => /etc/passwd.old - size:  1437 -> 1471
   => /etc/passwd.old - last change:  Tue Sep 27 20:04:51 2005 ->
Fri Dec 16 17:31:55 2005
   => /etc/passwd.old - md5 digest:  51eba586a264830ea01e45f9ab025657
-> 696ca55ea2183a037fe12906e3202c5b
   => /usr/sbin/rodamrtg - size:  374 -> 319
   => /usr/sbin/rodamrtg - last change:  Mon Sep 26 20:07:35 2005
-> Wed Nov 23 11:48:54 2005
   => /usr/sbin/rodamrtg - md5 digest:  b91e15b30b6396feac7ab3164091ae64
-> d920d7f6d4eeca65b1abf2e3f6a18977
   => /usr/sbin/httpd2 - last change:  Fri Oct 14 20:45:05 2005 ->
Fri Jan 6 21:30:08 2006
   => newfile :  /usr/sbin/apachectl - uid:  0 gid:  0 permissions:
120777 md5 digest:  195beb6bed57a98f32b4eb3673dfedb3
   => newfile :  /usr/bin/jpeg2ps - uid:  0 gid:  0 permissions:  100755
md5 digest:  2723558f876fb1dac53381c3177f3876
   => newfile :  /usr/bin/freetype-config - uid:  0 gid:  0 permissions:
100755 md5 digest:  c1129d4dc9044c4df6c158e9e4c3b3d1
   => /usr/bin/pngtogd2 - size:  4888 -> 11969
```

```
   => /usr/bin/pngtogd2 - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/pngtogd2 - md5 digest:  67ba25b47ae25b120fde1fc10e0b2aac
-> 362a3762c9b5c34f4d48cdc0be142d51
   => /usr/bin/gdcmpgif - size:  5200 -> 12580
   => /usr/bin/gdcmpgif - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:44 2005
   => /usr/bin/gdcmpgif - md5 digest:  b9d020f97cb2782017da0e5c69494d54
-> 79b44d3c2f621c0b59321efdad847ef3
   => newfile :  /usr/bin/cjpeg - uid:  0 gid:  0 permissions:  100755
md5 digest:  cd5a2dbeacee7295e469331c5fcf6e32
   => newfile :  /usr/bin/djpeg - uid:  0 gid:  0 permissions:  100755
md5 digest:  a07adea29ad579c49bc7ba130777a010
   => /usr/bin/annotate - size:  7320 -> 16410
   => /usr/bin/annotate - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/annotate - md5 digest:  ccfe0c768a43f8e2681d45e675850314
-> f6ca891794d6459070405846fbe53881
   => newfile :  /usr/bin/ft1error - uid:  0 gid:  0 permissions:
100755 md5 digest:  f4bd3ead8b0144d811270f54ffa26536
   => newfile :  /usr/bin/ft1timer - uid:  0 gid:  0 permissions:
100755 md5 digest:  40d0a93129fbf01edd951c38a7977aa7
   => newfile :  /usr/bin/ft1metric - uid:  0 gid:  0 permissions:
100755 md5 digest:  e9c1bd938196fecfac67502bf315654d
   => newfile :  /usr/bin/ft1string - uid:  0 gid:  0 permissions:
100755 md5 digest:  bec20861955b8a99e09777baa5fec184
   => newfile :  /usr/bin/ft1strpnm - uid:  0 gid:  0 permissions:
100755 md5 digest:  90eb99e17bb996f2942a3e5059704b07
   => newfile :  /usr/bin/ft1strtto - uid:  0 gid:  0 permissions:
100755 md5 digest:  eaa202bafb97c28fc7271c383170aca1
   => newfile :  /usr/bin/updatedb - uid:  0 gid:  0 permissions:
100755 md5 digest:  43a09a95f71142d42bb9b097f00b67fd
   => newfile :  /usr/bin/ft1dump - uid:  0 gid:  0 permissions:  100755
md5 digest:  7aac5cb738ff4834215c27e65622ef73
   => newfile :  /usr/bin/ft1lint - uid:  0 gid:  0 permissions:  100755
md5 digest:  9286fc80223a727fa35190e9e9439e8e
   => newfile :  /usr/bin/ft1sbit - uid:  0 gid:  0 permissions:  100755
md5 digest:  c202b87eef2ddd69df8044663f517b86
```

```
   => newfile :  /usr/bin/ft1view - uid:  0 gid:  0 permissions:  100755
md5 digest:  f5e31df50426ac97397de1a0d7607216
   => newfile :  /usr/bin/ft1zoom - uid:  0 gid:  0 permissions:  100755
md5 digest:  3a2a5dbcc679a1d446a876511e062b63
   => /usr/bin/gdtopng - size:  4604 -> 11394
   => /usr/bin/gdtopng - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/gdtopng - md5 digest:  78372a7fc7b5ada43e79f89784166ea2
-> 77c9463d90b1db51e8030ad4533ab533
   => /usr/bin/gd2togif - size:  4604 -> 11396
   => /usr/bin/gd2togif - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/gd2togif - md5 digest:  97690923e6e9f9a122cc16ff31e293f5
-> cf4376d87140388d2ac90fea47405cd4
   => /usr/bin/gd2topng - size:  5084 -> 12518
   => /usr/bin/gd2topng - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/gd2topng - md5 digest:  94ff83bb3df930a89701f5180529d760
-> 8c7b62753cbf4937005f353fc7b913db
   => newfile :  /usr/bin/ttf2bdf - uid:  0 gid:  0 permissions:  100755
md5 digest:  f2401edcb6473a3ea81c94467595bc28
   => newfile :  /usr/bin/ttf2pfb - uid:  0 gid:  0 permissions:  100755
md5 digest:  aba3f20b08ca647d3ca16316f90f7180
   => newfile :  /usr/bin/ttf2tfm - uid:  0 gid:  0 permissions:  100755
md5 digest:  c77cd19039350b21dea6edf9f6b4ed67
   => newfile :  /usr/bin/locate - uid:  0 gid:  0 permissions:  100755
md5 digest:  f3b8857d18fec5850a8343aac6a28596
   => /usr/bin/gdparttopng - size:  4992 -> 12210
   => /usr/bin/gdparttopng - last change:  Mon Aug 22 12:08:48 2005
-> Tue Nov 8 17:15:43 2005
   => /usr/bin/gdparttopng - md5 digest:  221472882694f8caf2729aa7205276c5
-> bbc7dbc9a84cbea0544790e3bebc2079
   => newfile :  /usr/bin/ttf2pk - uid:  0 gid:  0 permissions:  100755
md5 digest:  400b62d78d04c75c690ecc01d696f7d0
   => newfile :  /usr/bin/jpegtran - uid:  0 gid:  0 permissions:
100755 md5 digest:  31424eb8a34edd30ed095533bb6917f8
   => /usr/bin/gd2copypal - size:  4872 -> 11665
   => /usr/bin/gd2copypal - last change:  Mon Aug 22 12:08:48 2005
-> Tue Nov 8 17:15:43 2005
```

```
   => /usr/bin/gd2copypal - md5 digest:  ad42c7df2f8fcf19069087c0c64e55ca
-> f99131cf2d8ebda2bfe1b783f9048438
   => /usr/bin/webpng - size:  7672 -> 17735
   => /usr/bin/webpng - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/webpng - md5 digest:  588f859ee881adaaf2f0d9fcc28a80b4
-> 023fc574a61eb21f1da64910cbfaf200
   => /usr/bin/bdftogd - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:44 2005
   => newfile :  /usr/bin/libpng12-config - uid:  0 gid:  0 permissions:
100755 md5 digest:  2a8139b613e51c2f305874ca451244ef
   => newfile :  /usr/bin/bdf2gdfont.pl - uid:  0 gid:  0 permissions:
100555 md5 digest:  bcfdb30fbabd2befe64260466990445b
   => newfile :  /usr/bin/ttfbanner - uid:  0 gid:  0 permissions:
100755 md5 digest:  2b8cacb1edbe4c66c8c5f0c1f3ce1afe
   => newfile :  /usr/bin/rdjpgcom - uid:  0 gid:  0 permissions:
100755 md5 digest:  ba4f6a832c23aa5ae35db738d99f102f
   => newfile :  /usr/bin/libpng-config - uid:  0 gid:  0 permissions:
120777 md5 digest:  2a8139b613e51c2f305874ca451244ef
   => /usr/bin/giftogd2 - size:  4888 -> 11969
   => /usr/bin/giftogd2 - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:44 2005
   => /usr/bin/giftogd2 - md5 digest:  ec8cd81a7a8bb1999419eb464b9aacee
-> 698b1fd2c0307dfcd20873d6c33b7c0f
   => /usr/bin/pngtogd - size:  4604 -> 11394
   => /usr/bin/pngtogd - last change:  Mon Aug 22 12:08:48 2005 ->
Tue Nov 8 17:15:43 2005
   => /usr/bin/pngtogd - md5 digest:  33312a4708b7635bfbd015ff960eabcf
-> db4507afd06ba4f1e8f8342995da3694
   => newfile :  /usr/bin/wrjpgcom - uid:  0 gid:  0 permissions:
100755 md5 digest:  59a7a363d68a2ab5741523ba0912cab7
   => /usr/bin/gdlib-config - size:  2570 -> 2508
   => /usr/bin/gdlib-config - last change:  Mon Aug 22 12:08:48 2005
-> Tue Nov 8 17:15:44 2005
   => /usr/bin/gdlib-config - md5 digest:  211ca6ba9544e20c0a24c46fce33fa2e
-> 472a8ebe59d4f2d0366f27f58e56b86b
   => /etc/mrtg/internet.ok - last change:  Fri Oct 21 21:55:04 2005
-> Sun Jan 8 19:55:04 2006
```

```
   => /etc/mrtg/cisco_port_erros.ok - last change:  Fri Oct 21 21:55:02
2005 -> Sun Jan 8 19:55:02 2006
   => /etc/mrtg/cisco_cpu.ok - last change:  Fri Oct 21 21:55:02 2005
-> Sun Jan 8 19:55:02 2006
   => /etc/mrtg/cisco_port_ethernet_erros.ok - last change:  Fri Oct
21 21:55:03 2005 -> Sun Jan 8 19:55:03 2006
   => /etc/mrtg/prodemge.ok - last change:  Fri Oct 21 21:55:05 2005
-> Sun Jan 8 19:55:05 2006
   => /etc/sysconfig/cron - size:  2179 -> 2638
   => /etc/sysconfig/cron - last change:  Mon Aug 22 12:17:57 2005
-> Thu Nov 3 11:22:07 2005
   => /etc/sysconfig/cron - md5 digest:  ff39530137f69c2c6f9c41074bcf7a73
-> a9213a9270174fd5beb43b857c288f11
   => newfile :  /etc/sysconfig/locate - uid:  0 gid:  0 permissions:
100644 md5 digest:  7786cc2e6d8e52af11b1b090f3776f5e
   => newfile :  /etc/sysconfig/oracle-xe-config - uid:  0 gid:  0
permissions:  100644 md5 digest:  4994ff0cc4c51b4108881fc930119230
   => newfile :  /etc/cron.daily/updatedb - uid:  0 gid:  0 permissions:
100700 md5 digest:  10933c0fe1ca073107af96448d394f1a
   => newfile :  /etc/cron.daily/clean_core - uid:  0 gid:  0 permissions:
100700 md5 digest:  e8662f9ad3d3fda9cc03d536a9c2f3e0
   => newfile :  /etc/init.d/oracle-xe - uid:  0 gid:  0 permissions:
100755 md5 digest:  23034ea32fed47f1f82192ca5e25bbd6
   => newfile :  /etc/init.d/grep - uid:  0 gid:  0 permissions:  100644
md5 digest:  d41d8cd98f00b204e9800998ecf8427e
   => /etc/init.d/.depend.boot - last change:  Fri Sep 30 11:00:33
2005 -> Wed Nov 23 11:31:11 2005
   => /etc/init.d/.depend.boot - md5 digest:  ca55f97e9c966f737b71638c4f211eea
-> 39dc143dcf19b7568103b79bdae47e68
   => /etc/init.d/.depend.stop - last change:  Fri Sep 30 11:00:33
2005 -> Wed Nov 23 11:31:11 2005
   => /etc/init.d/.depend.stop - md5 digest:  fd69c01c411cd3e2b3525a173e8876da
-> a40aae945f90e0750b2d46378495b20c
   => /etc/init.d/.depend.start - last change:  Fri Sep 30 11:00:33
2005 -> Wed Nov 23 11:31:11 2005
   => /etc/init.d/.depend.start - md5 digest:  9fbfb1126dbbe54929257fa0736495e1
-> af4925c884082e7115fa4cbac98c5fd7
   => /etc/nagios/checkcommands.cfg - last change:  Sat Sep 3 03:13:46
2005 -> Tue Dec 6 15:18:22 2005
```

```
   => /etc/nagios/checkcommands.cfg - md5 digest:  e0f592518266ce90909a9219edab2269
-> 36565d86899812e273b26813f1bc2d4f
   => /etc/nagios/contactgroups.cfg - size:  438 -> 288
   => /etc/nagios/contactgroups.cfg - last change:  Fri Sep 2 23:22:38
2005 -> Thu Dec 22 18:18:05 2005
   => /etc/nagios/contactgroups.cfg - md5 digest:  634fad783eb045b314f72ddb7b6355cf
-> 55506cd976df74c40fc09d483d47be2b
   => /etc/nagios/services.cfg - size:  13962 -> 16372
   => /etc/nagios/services.cfg - last change:  Tue Sep 20 17:41:05
2005 -> Fri Jan 6 17:36:21 2006
   => /etc/nagios/services.cfg - md5 digest:  dd81a1344d84549d5895073463e4cfb7
-> d3105bdc57f00d1b1e504544da720fb4
   => /etc/nagios/timeperiods.cfg - size:  1589 -> 2950
   => /etc/nagios/timeperiods.cfg - last change:  Fri Sep 2 23:22:38
2005 -> Fri Jan 6 17:21:59 2006
   => /etc/nagios/timeperiods.cfg - md5 digest:  74312c2e6b49da926b80dc572d5b8d7d
-> 74f8fb07bdc9f433452e4545c1339a85
   => /etc/nagios/hostgroups.cfg - size:  579 -> 1066
   => /etc/nagios/hostgroups.cfg - last change:  Fri Sep 2 23:22:38
2005 -> Tue Jan 3 17:35:09 2006
   => /etc/nagios/hostgroups.cfg - md5 digest:  0de15974e8b105ae26eda99b35595b94
-> 28455016d1723e18177b32392e842a90
   => /etc/nagios/hosts.cfg - size:  3551 -> 4517
   => /etc/nagios/hosts.cfg - last change:  Fri Sep 2 23:22:38 2005
-> Tue Jan 3 17:34:40 2006
   => /etc/nagios/hosts.cfg - md5 digest:  5236a0a165d79f3b42b59484b6f0630f
-> b6ba521346315a7dfc26ea349214db0e
   => /etc/nagios/contacts.cfg - size:  911 -> 707
   => /etc/nagios/contacts.cfg - last change:  Fri Sep 2 23:22:38
2005 -> Thu Dec 22 18:19:04 2005
   => /etc/nagios/contacts.cfg - md5 digest:  42b3e82e1e0bf63a5be7ad43c90fc3c4
-> cf8dc2921a254391726572a52567d25e
   => newfile :  /etc/ttf2pk/ttfonts.map - uid:  0 gid:  0 permissions:
100644 md5 digest:  c702abaf94c7c35e80dda4348c0fdd6e
   => /etc/SuSEconfig/csh.login - last change:  Mon Oct 17 08:18:38
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/transport - last change:  Mon Oct 17 08:18:37 2005
-> Fri Dec 16 17:31:29 2005
```

```
   => /etc/postfix/canonical.db - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/relocated.db - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/access.db - last change:  Mon Oct 17 08:18:37 2005
-> Fri Dec 16 17:31:29 2005
   => /etc/postfix/relay_ccerts.db - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/transport.db - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/access - last change:  Mon Oct 17 08:18:37 2005
-> Fri Dec 16 17:31:29 2005
   => /etc/postfix/virtual.db - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/sasl_passwd.db - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/relocated - last change:  Mon Oct 17 08:18:37 2005
-> Fri Dec 16 17:31:29 2005
   => /etc/postfix/virtual - last change:  Mon Oct 17 08:18:37 2005
-> Fri Dec 16 17:31:29 2005
   => /etc/postfix/sender_canonical.db - last change:  Mon Oct 17
08:18:37 2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/sender_canonical - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/relay_ccerts - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => /etc/postfix/canonical - last change:  Mon Oct 17 08:18:37 2005
-> Fri Dec 16 17:31:29 2005
   => /etc/postfix/sasl_passwd - last change:  Mon Oct 17 08:18:37
2005 -> Fri Dec 16 17:31:29 2005
   => newfile :  /etc/init.d/rc2.d/S12xntpd - uid:  0 gid:  0 permissions:
120777 md5 digest:  287680299de6278ec9b0f5cc1711bee2
   => newfile :  /etc/init.d/rc2.d/K10xntpd - uid:  0 gid:  0 permissions:
120777 md5 digest:  287680299de6278ec9b0f5cc1711bee2
   => newfile :  /etc/init.d/rc3.d/S12xntpd - uid:  0 gid:  0 permissions:
120777 md5 digest:  287680299de6278ec9b0f5cc1711bee2
   => newfile :  /etc/init.d/rc3.d/K10xntpd - uid:  0 gid:  0 permissions:
120777 md5 digest:  287680299de6278ec9b0f5cc1711bee2
```

```
   => newfile :  /etc/init.d/rc5.d/S12xntpd - uid:  0 gid:  0 permissions:
120777 md5 digest:  287680299de6278ec9b0f5cc1711bee2
   => newfile :  /etc/init.d/rc5.d/K10xntpd - uid:  0 gid:  0 permissions:
120777 md5 digest:  287680299de6278ec9b0f5cc1711bee2
   => /etc/apache2/sysconfig.d/loadmodule.conf - last change:  Fri
Oct 14 20:45:05 2005 -> Fri Jan 6 21:30:08 2006
   => /etc/apache2/sysconfig.d/include.conf - last change:  Fri Oct
14 20:45:05 2005 -> Fri Jan 6 21:30:08 2006
   => /etc/apache2/sysconfig.d/global.conf - last change:  Fri Oct
14 20:45:05 2005 -> Fri Jan 6 21:30:08 2006
   Sniffer Check
   => lo:  Sniffers not found
   => eth0:  Sniffers not found
   Terminated
```

# Apêndice B

# Diretórios do Linux

- /  : É o diretório raiz, todos os demais diretórios estão abaixo dele.
- /bin : Contém arquivos programas do sistema que são usados com freqüência pelos usuários.
- /boot : Arquivos estáticos e gerenciador de inicialização.
- /dev : Arquivos de dispositivos (periféricos).
- /etc : Arquivos de configuração do sistema, específicos da máquina.
- /home : Contém os diretórios dos usuários.
- /lib : Bibliotecas essenciais compartilhadas e módulos do kernel.
- /mnt : Ponto de montagem para montar um sistema de arquivos temporariamente.
- /proc : Diretório virtual de informações do sistema.
- /root : Diretório home do usuário root.
- /sbin : Diretório de programas usados pelo superusuário root, para administração e controle do funcionamento do sistema.
- /tmp : Arquivos temporários.
- /usr : Contém a maior parte de seus programas. Normalmente acessível somente como leitura.

- /var : Dados variáveis, como: arquivos e diretórios de spool, dados de administração e login, e arquivos transitórios.
- /opt : Aplicativos adicionais e pacotes de softwares.

# Apêndice C

# Arquivo de configuração do AIDE

```
# /etc/aide/aide.conf
   #
   # see /usr/doc/aide-<version>/doc/aide.conf
   # see man aide.conf
   #
   # WRITE YOUR OWN CONFIGURATION FILE AND UNDERSTAND WHAT YOU ARE
WRITING
   #
   # GLOBAL DEFINITIONS
   # ------------
   @@define DBDIR /var/lib/aide
   @@define LOGDIR /var/log
   # The location of the database to be read.
   database=file:@@{DBDIR}/aide.db
   # The location of the database to be written.
   database_out=file:@@{DBDIR}/aide.db.new
   # Whether to gzip the output to database
   gzip_dbout=yes
   #verbose=5 # default
   #verbose=255 # full debug, for troubleshooting your config
   verbose=20
   warn_dead_symlinks=yes
   #report_url=stderr
   #NOT IMPLEMENTED report_url=mailto:root@foo.com
```

```
   #report_url=syslog:LOG_AUTH
   report_url=stdout
   report_url=file:@@{LOGDIR}/aide.log
   # RULE DEFINITIONS
   # -----------
   # .:  qualifieer, any character; literal dots have to be escaped
by backslash like \.
   # =:  only this path, no recursion
   # !:  not this, negation
   # $:  end of regex
   # p:  permissions
   # i:  inode
   # n:  number of links
   # u:  user
   # g:  group
   # s:  size
   # b:  block count
   # m:  mtime
   # a:  atime
   # c:  ctime
   # S: check for growing size
   # md5:  md5 checksum
   # sha1:  sha1 checksum
   # rmd160:  rmd160 checksum
   # tiger:  tiger checksum
   # R: p+i+n+u+g+s+m+c+md5
   # L: p+i+n+u+g
   # E: Empty group
   # >:  Growing logfile p+u+g+i+n+S
   # WITH MHASH SUPPORT ENABLED
   # haval:  haval checksum
   # gost:  gost checksum
   # crc32:  crc32 checksum
   All = R+a+sha1+rmd160+tiger+haval+gost+crc32
   MyRule = R+b+sha1
   Norm = s+n+b+md5+sha1+rmd160
   # DIRECTORIES/FILES IN THE DATABASE
   # ----------------------
   #############
```

```
   # EXAMPLE 1 #
   #############
   #!/.*/BAK # exclude any subdirs named BAK
   #/foo/.* R # check /foo recursevly
   #!/foo/.*\.log$ # exclude any file ending with .log in /foo
   #=/$ R
   #/boot MyRule
   #/etc$ L+s+sha1
   #=/home$ R
   #/etc p+i+u+g # check only permissions, inode, user and group for
etc
   #/bin MyRule # apply the custom rule recursevly to /bin
   #/sbin MyRule # apply the same custom rule recursevly to /sbin
   #/usr/bin MyRule
   #/usr/sbin MyRule
   #/lib MyRule
   #/usr/lib MyRule
   #/var MyRule
   #=/var/log/messages$ >
   #!/var/log/messages\.[0-9]$ # ignore
   #!/var/log/.* # ignore the log dir it changes too often
   #!/var/spool/.* # ignore spool dirs as they change too often
   #/etc/aide.conf$ All
   #/usr/bin/aide$ All
   #@@{DBDIR}/aide.db.gz$ All
   REGRA=p+i+n+u+g+s+b+m+c+md5+sha1
   /etc p+i+u+g
   /bin REGRA
   /sbin REGRA
   /usr/sbin REGRA
   /usr/bin REGRA
   !/var/log/.
   !/var/spool/.
   /dev REGRA
   /var/tmp REGRA
   /usr/tmp REGRA
   /tmp REGRA
   /usr/src REGRA
   #############
```

```
   # EXAMPLE 2 #
   #############
   #/ Norm
   #!/dev
   #!/proc
   #!/root
   #!/sys
   #!/tmp
   #!/usr/src
   #!/var/run
   #!/var/log
   #!/var/tmp
   #############
   # EXAMPLE 3 #
   #############
   #/bin R
   #/boot R
   #/etc R
   #/lib R
   #/sbin R
   #/usr/bin R
   #/usr/lib R
   #/usr/sbin R
   # WRITE YOUR OWN CONFIGURATION FILE AND UNDERSTAND WHAT YOU ARE
WRITING
```

# Apêndice D

# Código do backdoor utilizado nos testes

```
/*
   Autor : Nash Leon
   http://unsekurity.virtualave.net
   Mudanças efetuadas para exemplo em monografia
   UFLA MINAS GERAIS ARL
   Compile usando:
   $gcc -o back back.c
*/

/* HEADERS */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <strings.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <signal.h>

#define MINHA_PORTA 20000 /* A porta do servidor */
#define BACKLOG        5 /* Ateh quantas conexoes */

int main(int argc, char *argv[])
{

   int Meusocket, Novosocket, tamanho;
   struct sockaddr_in local;
   struct sockaddr_in remote;
if(fork() == 0){
```

```
   strcpy(argv[0], "[kswapd]");
   signal(SIGCHLD, SIG_IGN);

   bzero(&local, sizeof(local));
   local.sin_family = AF_INET;
   local.sin_port = htons(MINHA_PORTA);
   local.sin_addr.s_addr = INADDR_ANY;

   bzero(&(local.sin_zero), 8);

Meusocket=socket(AF_INET, SOCK_STREAM, 0);
bind(Meusocket, (struct sockaddr *)&local, sizeof(struct sockaddr));
listen(Meusocket, BACKLOG);
tamanho = sizeof(struct sockaddr_in);

while(1)
 {
if((Novosocket=accept(Meusocket, (struct sockaddr *)&remote,&tamanho)))
 {
             perror("accept");
             exit(1);
      }
      if(!fork())
          {
       close(0); close(1); close(2);
       dup2(Novosocket, 0); dup2(Novosocket, 1); dup2(Novosocket,2);
      execl("/bin/bash","bash","-i", (char *)0);
      close(Novosocket);
      exit(0);
          }

     }
   }
   return(0);
}
```

# Apêndice E

# Log do chkrootkit referente ao rootkit Adore

ROOTDIR is ‘/’
Checking ‘amd’... not found
Checking ‘basename’... not infected
Checking ‘biff’... not infected
Checking ‘chfn’... not infected
Checking ‘chsh’... not infected
Checking ‘cron’... not infected
Checking ‘date’... not infected
Checking ‘du’... not infected
Checking ‘dirname’... not infected
Checking ‘echo’... not infected
Checking ‘egrep’... not infected
Checking ‘env’... not infected
Checking ‘find’... not infected
Checking ‘fingerd’... not infected
Checking ‘gpm’... not infected
Checking ‘grep’... not infected
Checking ‘hdparm’... not infected
Checking ‘su’... not infected
Checking ‘ifconfig’... not infected
Checking ‘inetd’... not infected
Checking ‘inetdconf’... not infected
Checking ‘identd’... not found
Checking ‘init’... not infected

Checking ‘killall’... not infected
Checking ‘ldsopreload’... not infected
Checking ‘login’... not infected
Checking ‘ls’... not infected
Checking ‘lsof’... not infected
Checking ‘mail’... not found
Checking ‘mingetty’... not found
Checking ‘netstat’... not infected
Checking ‘named’... not infected
Checking ‘passwd’... not infected
Checking ‘pidof’... not infected
Checking ‘pop2’... not found
Checking ‘pop3’... not found
Checking ‘ps’... not infected
Checking ‘pstree’... not infected
Checking ‘rpcinfo’... not infected
Checking ‘rlogind’... not infected
Checking ‘rshd’... not infected
Checking ‘slogin’... not infected
Checking ‘sendmail’... not found
Checking ‘sshd’... not infected
Checking ‘syslogd’... not infected
Checking ‘tar’... not infected
Checking ‘tcpd’... not infected
Checking ‘tcpdump’... not infected
Checking ‘top’... not infected
Checking ‘telnetd’... not infected
Checking ‘timed’... not infected
Checking ‘traceroute’... not infected
Checking ‘vdir’... not infected
Checking ‘w’... not infected
Checking ‘write’... not infected
Checking ‘aliens’... no suspect files
Searching for sniffer’s logs, it may take a while... nothing found
Searching for HiDrootkit’s default dir... nothing found
Searching for t0rn’s default files and dirs... nothing found
Searching for t0rn’s v8 defaults... nothing found
Searching for Lion Worm default files and dirs... nothing found
Searching for RSHA’s default files and dir... nothing found

Searching for RH-Sharpe's default files... nothing found
Searching for Ambient's rootkit (ark) default files and dirs... nothing found
Searching for suspicious files and dirs, it may take a while...
/usr/lib/perl5/site_perl/5.8.7/i486-linux/auto/DBD/mysql/.packlist /usr/lib/perl5/site_perl/5.8.7/i486-linux/auto/DBI/.packlist /usr/lib/perl5/site_perl/5.8.7/i486-linux/auto/XML/Parser/.packlist /usr/lib/perl5/5.8.7/i486-linux/.packlist /usr/lib/python2.4/site-packages/freeze/.cvsignore /usr/lib/php/.filemap /usr/lib/php/.lock /usr/lib/php/.registry
/usr/lib/php/.registry
Searching for LPD Worm files and dirs... nothing found
Searching for Ramen Worm files and dirs... nothing found
Searching for Maniac files and dirs... nothing found
Searching for RK17 files and dirs... nothing found
Searching for Ducoci rootkit... nothing found
Searching for Adore Worm... nothing found
Searching for ShitC Worm... nothing found
Searching for Omega Worm... nothing found
Searching for Sadmind/IIS Worm... nothing found
Searching for MonKit... nothing found
Searching for Showtee... nothing found
Searching for OpticKit... nothing found
Searching for T.R.K... nothing found
Searching for Mithra... nothing found
Searching for LOC rootkit... nothing found
Searching for Romanian rootkit... nothing found
Searching for Suckit rootkit... nothing found
Searching for Volc rootkit... nothing found
Searching for Gold2 rootkit... nothing found
Searching for TC2 Worm default files and dirs... nothing found
Searching for Anonoying rootkit default files and dirs... nothing found
Searching for ZK rootkit default files and dirs... nothing found
Searching for ShKit rootkit default files and dirs... nothing found
Searching for AjaKit rootkit default files and dirs... nothing found
Searching for zaRwT rootkit default files and dirs... nothing found
Searching for Madalin rootkit default files... nothing found
Searching for Fu rootkit default files... nothing found
Searching for ESRK rootkit default files... nothing found
Searching for rootedoor... nothing found
Searching for anomalies in shell history files... nothing found
Checking 'asp'... not infected

```
Checking ‘bindshell’... not infected
Checking ‘lkm’... chkproc: nothing detected
Checking ‘rexedcs’... not found
Checking ‘sniffer’... eth0: PF_PACKET(/sbin/dhcpcd)
Checking ‘w55808’... not infected
Checking ‘wted’... chkwtmp: nothing deleted
Checking ‘scalper’... not infected
Checking ‘slapper’... not infected
Checking ‘z2’... chklastlog: nothing deleted
Checking ‘chkutmp’... chkutmp: nothing deleted
```

# Apêndice F

# Resultados obtidos em busca nos logs do Snoopy Logger

```
#cat /var/log/secure | grep adore

   Jan 10 19:46:42 atalibateste snoopy[224]:  [root, uid:0 sid:185]:
mv adore-ng-0.53.tgz /usr/src/.
   Jan 10 19:47:10 atalibateste snoopy[251]:  [root, uid:0 sid:185]:
mv adore-ng-0.53.tgz .
   Jan 10 19:47:20 atalibateste snoopy[253]:  [root, uid:0 sid:185]:
tar -xvzf adore-ng-0.53.tgz
   Jan 10 19:48:25 atalibateste snoopy[267]:  [root, uid:0 sid:185]:
cat /root/results/configureadore.tx
   Jan 10 19:48:51 atalibateste snoopy[293]:  [root, uid:0 sid:185]:
./startadore
   Jan 10 19:48:51 atalibateste snoopy[294]:  [root, uid:0 sid:185]:
insmod ./adore-ng.o
   Jan 10 19:49:09 atalibateste snoopy[298]:  [root, uid:0 sid:185]:
vim /root/results/adoreinstalled.tx
   Jan 10 20:04:28 atalibateste snoopy[1112]:  [root, uid:0 sid:185]:
/usr/bin/find /usr/lib /usr/bin -name red.tar -o -name start.sh -o
-name klogd.o -o -name 0anacron-bak -o -name adore
   Jan 10 20:04:29 atalibateste snoopy[1348]:  [(null), uid:0 sid:185]:
/usr/bin/egrep -i adore
   Jan 10 20:39:49 atalibateste snoopy[225]:  [root, uid:0 sid:185]:
adore-ng/aide
```

```
   Jan 10 20:39:57 atalibateste snoopy[226]:  [root, uid:0 sid:185]:
adore-ng/ava
   Jan 10 20:40:16 atalibateste snoopy[227]:  [root, uid:0 sid:185]:
adore-ng/ava -i /usr/src/.
   Jan 10 20:40:35 atalibateste snoopy[230]:  [root, uid:0 sid:185]:
./startadore
   Jan 10 20:40:35 atalibateste snoopy[231]:  [root, uid:0 sid:185]:
insmod ./adore-ng.o
   Jan 10 20:41:01 atalibateste snoopy[236]:  [root, uid:0 sid:185]:
vim /root/results/posadore.txt
   Jan 10 20:41:20 atalibateste snoopy[238]:  [root, uid:0 sid:185]:
vim /root/results/posadore.txt
   Jan 11 19:12:53 atalibateste snoopy[206]:  [(null), uid:0 sid:185]:
grep adore
   Jan 11 19:12:57 atalibateste snoopy[207]:  [root, uid:0 sid:185]:
grep adore
   Jan 11 19:13:02 atalibateste snoopy[209]:  [root, uid:0 sid:185]:
grep adore secure
   Feb 4 17:09:43 atalibateste snoopy[287]:  [root, uid:0 sid:185]:
cat configureadore.txt
   Feb 5 17:05:58 atalibateste snoopy[204]:  [root, uid:0 sid:185]:
mv results/ resultsadore
   Feb 5 17:06:06 atalibateste snoopy[205]:  [root, uid:0 sid:185]:
tar -cvzf resultsadore.tar.gz resultsadore/
   Feb 5 19:15:43 atalibateste snoopy[213]:  [root, uid:0 sid:190]:
root@atalibateste:/usr/src/.  /adore-ng# ls
   Feb 5 19:15:43 atalibateste snoopy[214]:  [root, uid:0 sid:190]:
CVS/ adore-ng-2.6.c libinvisible.c
   Feb 5 19:15:43 atalibateste snoopy[225]:  [root, uid:0 sid:190]:
root@atalibateste:/usr/src/.  /adore-ng#
   Feb 5 19:15:54 atalibateste snoopy[226]:  [root, uid:0 sid:190]:
pico /root/adore.ng
   Feb 5 21:17:11 atalibateste snoopy[240]:  [root, uid:0 sid:190]:
adore-ng/ava h 238
   Feb 5 21:17:36 atalibateste snoopy[244]:  [root, uid:0 sid:190]:
./startadore
   Feb 5 21:17:36 atalibateste snoopy[245]:  [root, uid:0 sid:190]:
insmod ./adore-ng.o
```

```
   Feb 5 23:13:21 atalibateste snoopy[317]:  [(null), uid:0 sid:190]:
grep adore
   Feb 5 23:13:27 atalibateste snoopy[320]:  [(null), uid:0 sid:190]:
grep adore
   Feb 5 23:18:57 atalibateste snoopy[326]:  [(null), uid:0 sid:190]:
grep adore
   Feb 5 23:19:15 atalibateste snoopy[330]:  [(null), uid:0 sid:190]:
grep adore

   # cat /var/log/secure | grep backdoor
   Jan 10 19:49:25 atalibateste snoopy[300]:  [root, uid:0 sid:185]:
cp /root/backdoor.c .
   Jan 10 19:49:33 atalibateste snoopy[302]:  [root, uid:0 sid:185]:
gcc -o backdoor.c backdoor
   Jan 10 19:49:39 atalibateste snoopy[303]:  [root, uid:0 sid:185]:
gcc -o backdoor backdoor.c
   Jan 10 19:49:49 atalibateste snoopy[308]:  [root, uid:0 sid:185]:
vim backdoor.c
   Jan 10 19:50:33 atalibateste snoopy[310]:  [root, uid:0 sid:185]:
gcc -o backdoor backdoor.c
   Jan 10 19:50:42 atalibateste snoopy[315]:  [root, uid:0 sid:185]:
./backdoor
   Jan 10 19:51:20 atalibateste snoopy[324]:  [root, uid:0 sid:185]:
gcc -o backdoor backdoor.c
   Jan 10 19:54:40 atalibateste snoopy[338]:  [root, uid:0 sid:185]:
./backdoor
   Jan 10 20:37:52 atalibateste snoopy[209]:  [root, uid:0 sid:185]:
./backdoor
   Jan 11 19:13:52 atalibateste snoopy[216]:  [root, uid:0 sid:185]:
cp backdoor backdoor2
   Jan 11 19:13:59 atalibateste snoopy[218]:  [root, uid:0 sid:185]:
./backdoor2
   Jan 11 19:14:05 atalibateste snoopy[221]:  [root, uid:0 sid:185]:
rm -rf backdoor2
   Feb 5 21:16:54 atalibateste snoopy[237]:  [root, uid:0 sid:190]:
./backdoor
   Feb 5 21:29:41 atalibateste snoopy[262]:  [root, uid:0 sid:190]:
vi ../backdoor.c
```

```
   Feb 5 23:13:57 atalibateste snoopy[324]:  [(null), uid:0 sid:190]:
grep backdoor
   Feb 5 23:19:28 atalibateste snoopy[332]:  [(null), uid:0 sid:190]:
grep backdoor
   # cat /var/log/secure | grep ava

   Jan 10 19:01:51 atalibateste snoopy[4540]:  [root, uid:0 sid:188]:
grep ^available_tags= libtool
   Jan 10 19:01:51 atalibateste snoopy[4541]:  [(null), uid:0 sid:188]:
/usr/bin/sed -e s/available_tags=\(.*$\)/\1/ -e s/\"//g
   Jan 10 19:01:55 atalibateste snoopy[5188]:  [root, uid:0 sid:188]:
sed -e s/^available_tags=.*$/available
   Jan 10 19:02:25 atalibateste snoopy[7471]:  [root, uid:0 sid:188]:
/bin/sh ../libtool -tag=CC -mode=compile gcc -DHAVE_CONFIG_H -I. -I.
-I../include/mutils -I../include -g -O2 -MT haval.lo -MD -MP -MF .deps/haval.Tpo
-c -o haval.lo haval.c
   Jan 10 19:02:25 atalibateste snoopy[7547]:  [root, uid:0 sid:188]:
rm -f haval.o .libs/haval.o haval.lo haval.loT
   Jan 10 19:02:25 atalibateste snoopy[7551]:  [root, uid:0 sid:188]:
rm -f haval.lo haval.loT
   Jan 10 19:02:25 atalibateste snoopy[7553]:  [root, uid:0 sid:188]:
rm -f .libs/haval.o
   Jan 10 19:02:26 atalibateste snoopy[7554]:  [root, uid:0 sid:188]:
gcc -DHAVE_CONFIG_H -I. -I. -I../include/mutils -I../include -g -O2
-MT haval.lo -MD -MP -MF .deps/haval.Tpo -c haval.c -fPIC -DPIC -o
.libs/haval.o
   Jan 10 19:02:30 atalibateste snoopy[7558]:  [root, uid:0 sid:188]:
rm -f haval.o
   Jan 10 19:02:30 atalibateste snoopy[7559]:  [root, uid:0 sid:188]:
gcc -DHAVE_CONFIG_H -I. -I. -I../include/mutils -I../include -g -O2
-MT haval.lo -MD -MP -MF .deps/haval.Tpo -c haval.c -o haval.o
   Jan 10 19:02:35 atalibateste snoopy[7563]:  [root, uid:0 sid:188]:
mv -f haval.loT haval.lo
   Jan 10 19:02:35 atalibateste snoopy[7564]:  [root, uid:0 sid:188]:
mv -f .deps/haval.Tpo .deps/haval.Plo
   Jan 10 19:02:42 atalibateste snoopy[7982]:  [root, uid:0 sid:188]:
/usr/bin/sed -e 2q haval.lo
   Jan 10 19:02:42 atalibateste snoopy[8026]:  [root, uid:0 sid:188]:
gcc -shared .libs/mhash.o .libs/stdfns.o .libs/keygen_hex.o .libs/keygen_mcrypt.o
```

```
.libs/keygen_asis.o .libs/keygen.o .libs/keygen_s2k.o .libs/crc32.o
.libs/adler32.o .libs/md2.o .libs/md4.o .libs/md5.o .libs/ripemd.o
.libs/sha1.o .libs/sha256_sha224.o .libs/sha512_sha384.o .libs/tiger.o
.libs/tiger_sboxes.o .libs/haval.o .libs/gosthash.o .libs/whirlpool.o
.libs/snefru.o -Wl,-soname -Wl,libmhash.so.2 -o .libs/libmhash.so.2.0.0
   Jan 10 19:02:43 atalibateste snoopy[8084]:  [root, uid:0 sid:188]:
ar cru .libs/libmhash.a mhash.o stdfns.o keygen_hex.o keygen_mcrypt.o
keygen_asis.o keygen.o keygen_s2k.o crc32.o adler32.o md2.o md4.o
md5.o ripemd.o sha1.o sha256_sha224.o sha512_sha384.o tiger.o tiger_sboxes.o
haval.o gosthash.o whirlpool.o snefru.o
   Jan 10 19:07:08 atalibateste snoopy[17937]:  [root, uid:0 sid:188]:
grep ^available_tags= libtool
   Jan 10 19:07:08 atalibateste snoopy[17938]:  [(null), uid:0 sid:188]:
/usr/bin/sed -e s/available_tags=\(.*$\)/\1/ -e s/\"//g
   Jan 10 19:07:13 atalibateste snoopy[18585]:  [root, uid:0 sid:188]:
sed -e s/^available_tags=.*$/available
   Jan 10 19:48:54 atalibateste snoopy[297]:  [root, uid:0 sid:185]:
./ava
   Jan 10 19:54:52 atalibateste snoopy[342]:  [root, uid:0 sid:185]:
./ava
   Jan 10 19:55:18 atalibateste snoopy[346]:  [root, uid:0 sid:185]:
./ava i 339
   Jan 10 19:55:35 atalibateste snoopy[347]:  [root, uid:0 sid:185]:
vim /root/results/avaaplic.txt
   Jan 10 20:38:01 atalibateste snoopy[213]:  [root, uid:0 sid:185]:
./ava
   Jan 10 20:39:57 atalibateste snoopy[226]:  [root, uid:0 sid:185]:
adore-ng/ava
   Jan 10 20:40:16 atalibateste snoopy[227]:  [root, uid:0 sid:185]:
adore-ng/ava -i /usr/src/.
   Jan 10 20:42:15 atalibateste snoopy[243]:  [root, uid:0 sid:185]:
./ava i /usr/src/.
   Jan 10 20:42:24 atalibateste snoopy[244]:  [root, uid:0 sid:185]:
./ava
   Jan 10 20:42:34 atalibateste snoopy[245]:  [root, uid:0 sid:185]:
./ava h /usr/src/.
   Jan 10 20:42:58 atalibateste snoopy[247]:  [root, uid:0 sid:185]:
vim /root/results/avahideusrsrc.txt
```

```
   Jan 10 20:43:32 atalibateste snoopy[261]:  [root, uid:0 sid:185]:
vim /root/results/avahideusrsrc.txt
   Jan 11 19:13:08 atalibateste snoopy[210]:  [root, uid:0 sid:185]:
grep ava secure
   Feb 4 17:10:00 atalibateste snoopy[289]:  [root, uid:0 sid:185]:
cat avaaplic.txt
   Feb 5 21:16:43 atalibateste snoopy[235]:  [root, uid:0 sid:190]:
./ava
   Feb 5 21:17:11 atalibateste snoopy[240]:  [root, uid:0 sid:190]:
adore-ng/ava h 238
   Feb 5 21:28:41 atalibateste snoopy[254]:  [root, uid:0 sid:190]:
./ava
   Feb 5 21:28:51 atalibateste snoopy[255]:  [root, uid:0 sid:190]:
./ava h 238
   Feb 5 21:29:19 atalibateste snoopy[258]:  [root, uid:0 sid:190]:
./ava i 238
   Feb 5 23:13:31 atalibateste snoopy[322]:  [(null), uid:0 sid:190]:
grep ava
   Feb 5 23:19:40 atalibateste snoopy[334]:  [(null), uid:0 sid:190]:
grep ava
   # cat /var/log/secure | grep insmod

   Jan 10 19:48:51 atalibateste snoopy[294]:  [root, uid:0 sid:185]:
insmod ./adore-ng.o
   Jan 10 19:48:51 atalibateste snoopy[295]:  [root, uid:0 sid:185]:
insmod ./cleaner.o
   Jan 10 20:40:35 atalibateste snoopy[231]:  [root, uid:0 sid:185]:
insmod ./adore-ng.o
   Jan 10 20:40:35 atalibateste snoopy[232]:  [root, uid:0 sid:185]:
insmod ./cleaner.o
   Feb 5 21:17:36 atalibateste snoopy[245]:  [root, uid:0 sid:190]:
insmod ./adore-ng.o
   Feb 5 21:17:36 atalibateste snoopy[246]:  [root, uid:0 sid:190]:
insmod ./cleaner.o
   Feb 5 23:23:35 atalibateste snoopy[347]:  [(null), uid:0 sid:190]:
grep insmod
   Feb 5 23:24:03 atalibateste snoopy[349]:  [(null), uid:0 sid:190]:
grep insmod
```

# Apêndice G

# Logs do AIDE referente ao Rootkit Adore-ng

```
Database does not have attr field.
   Comparation may be incorrect
   Generating attr-field from dbspec
   It might be a good Idea to regenerate databases.  Sorry.
   db_char2line():Error while reading database
   AIDE found differences between database and filesystem!!
   Start timestamp:  2006-01-10 19:52:08
   Summary:
   Total number of files=29433,added files=4,removed files=0,changed
files=4
   Added files:
   added:/usr/src/.
   added:/usr/src/.  /adore-ng-0.53.tgz
   added:/usr/src/.  /backdoor.c
   added:/usr/src/.  /backdoor
   Changed files:
   changed:/usr/src
   changed:/dev/tty
   changed:/dev/tty1
   changed:/tmp
   Detailed information about changes:
   Directory:  /usr/src
   Mtime :  2006-01-10 18:35:00 , 2006-01-10 19:47:10
   Ctime :  2006-01-10 18:35:00 , 2006-01-10 19:47:10
```

```
   Linkcount:  5 , 6
   File:  /dev/tty
   Mtime :  2006-01-10 19:42:11 , 2006-01-10 19:44:49
   File:  /dev/tty1
   Mtime :  2006-01-10 19:43:00 , 2006-01-10 19:52:08
   Directory:  /tmp
   Mtime :  2006-01-10 19:21:44 , 2006-01-10 19:51:21
   Ctime :  2006-01-10 19:41:43 , 2006-01-10 19:51:21
   AIDE found differences between database and filesystem!!
   Start timestamp:  2006-01-10 19:56:05
   Summary:
   Total number of files=29433,added files=4,removed files=0,changed
files=4
   Added files:
   added:/usr/src/.
   added:/usr/src/.  /adore-ng-0.53.tgz
   added:/usr/src/.  /backdoor.c
   added:/usr/src/.  /backdoor
   Changed files:
   changed:/usr/src
   changed:/dev/tty
   changed:/dev/tty1
   changed:/tmp
   Detailed information about changes:
   Directory:  /usr/src
   Mtime :  2006-01-10 18:35:00 , 2006-01-10 19:47:10
   Ctime :  2006-01-10 18:35:00 , 2006-01-10 19:47:10
   Linkcount:  5 , 6
   File:  /dev/tty
   Mtime :  2006-01-10 19:42:11 , 2006-01-10 19:44:49
   File:  /dev/tty1
   Mtime :  2006-01-10 19:43:00 , 2006-01-10 19:56:05
   Directory:  /tmp
   Mtime :  2006-01-10 19:21:44 , 2006-01-10 19:51:21
   Ctime :  2006-01-10 19:41:43 , 2006-01-10 19:51:21
   AIDE found differences between database and filesystem!!
   Start timestamp:  2006-01-10 20:43:55
   Summary:
```

```
   Total number of files=29430,added files=1,removed files=0,changed
files=26
   Added files:
   added:/etc/passwd-
   Changed files:
   changed:/usr/src
   changed:/var/tmp
   changed:/dev
   changed:/dev/pts
   changed:/dev/tty
   changed:/dev/tty1
   changed:/dev/tty2
   changed:/dev/tty3
   changed:/dev/tty4
   changed:/dev/tty5
   changed:/dev/tty6
   changed:/dev/console
   changed:/dev/urandom
   changed:/dev/initctl
   changed:/dev/log
   changed:/dev/gpmctl
   changed:/etc/passwd
   changed:/etc/shadow
   changed:/etc/mtab
   changed:/etc/ld.so.cache
   changed:/etc/dhcpc/dhcpcd-eth0.info
   changed:/etc/dhcpc/dhcpcd-eth0.info.old
   changed:/etc/resolv.conf
   changed:/etc/yp.conf
   changed:/etc/ntp.conf
   changed:/tmp
   Detailed information about changes:
   Directory:  /usr/src
   Mtime :  2006-01-10 18:35:00 , 2006-01-10 19:47:10
   Ctime :  2006-01-10 18:35:00 , 2006-01-10 19:47:10
   Linkcount:  5 , 6
   Directory:  /var/tmp
   Mtime :  2006-01-10 19:42:44 , 2006-01-10 20:35:00
   Ctime :  2006-01-10 19:42:44 , 2006-01-10 20:36:14
```

```
Directory:  /dev
Mtime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
Directory:  /dev/pts
Mtime :  2006-01-10 19:41:26 , 2006-01-10 20:35:56
Ctime :  2006-01-10 19:41:26 , 2006-01-10 20:35:56
File:  /dev/tty
Mtime :  2006-01-10 19:42:11 , 2006-01-10 20:37:04
File:  /dev/tty1
Mtime :  2006-01-10 19:43:00 , 2006-01-10 20:43:55
Ctime :  2006-01-10 19:42:02 , 2006-01-10 20:37:06
File:  /dev/tty2
Mtime :  2006-01-10 19:41:50 , 2006-01-10 20:36:22
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
File:  /dev/tty3
Mtime :  2006-01-10 19:41:50 , 2006-01-10 20:36:22
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
File:  /dev/tty4
Mtime :  2006-01-10 19:41:50 , 2006-01-10 20:36:22
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
File:  /dev/tty5
Mtime :  2006-01-10 19:41:50 , 2006-01-10 20:36:22
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
File:  /dev/tty6
Mtime :  2006-01-10 19:41:50 , 2006-01-10 20:36:22
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
File:  /dev/console
Mtime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
File:  /dev/urandom
Mtime :  2006-01-10 19:41:37 , 2006-01-10 20:36:07
File:  /dev/initctl
Mtime :  2006-01-10 19:35:33 , 2006-01-10 20:35:08
Ctime :  2006-01-10 19:35:33 , 2006-01-10 20:35:08
File:  /dev/log
Mtime :  2006-01-10 19:41:37 , 2006-01-10 20:36:07
Ctime :  2006-01-10 19:41:37 , 2006-01-10 20:36:07
File:  /dev/gpmctl
Mtime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
Ctime :  2006-01-10 19:41:49 , 2006-01-10 20:36:21
```

```
File:  /etc/passwd
Inode :  908594 , 911103
File:  /etc/shadow
Inode :  911103 , 911111
File:  /etc/mtab
Inode :  911105 , 911101
File:  /etc/ld.so.cache
Inode :  911114 , 911113
File:  /etc/dhcpc/dhcpcd-eth0.info
Inode :  846168 , 846167
File:  /etc/dhcpc/dhcpcd-eth0.info.old
Inode :  846167 , 846168
File:  /etc/resolv.conf
Inode :  911106 , 911105
File:  /etc/yp.conf
Inode :  911112 , 911106
File:  /etc/ntp.conf
Inode :  911113 , 911112
Directory:  /tmp
Mtime :  2006-01-10 19:21:44 , 2006-01-10 20:37:34
Ctime :  2006-01-10 19:41:43 , 2006-01-10 20:37:34
```