UNIVERSIDADE DO EXTREMO SUL CATARINENSE
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADAUTO DE SOUZA BERNARDO

TÉCNICAS COMPUTACIONAIS NO AUXÍLIO À PERÍCIA FORENSE NA ANÁLISE DE EVIDÊNCIAS COLETADAS EM SERVIDORES GNU/LINUX

CRICIÚMA, JUNHO DE 2006.

UNIVERSIDADE DO EXTREMO SUL CATARINENSE
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ADAUTO DE SOUZA BERNARDO

TÉCNICAS COMPUTACIONAIS NO AUXÍLIO À PERÍCIA FORENSE NA ANÁLISE DE EVIDÊNCIAS COLETADAS EM SERVIDORES GNU/LINUX

Trabalho de Conclusão de Curso para a Obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. M.Sc. Paulo Martins

CRICIÚMA, JUNHO DE 2006.

Adauto de Souza Bernardo

TÉCNICAS COMPUTACIONAIS NO AUXÍLIO À PERÍCIA FORENSE NA ANÁLISE DE EVIDÊNCIAS COLETADAS EM SERVIDORES GNU/LINUX

Submetido ao corpo docente do Departamento de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

_____

Prof. M.Sc. Ana Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

_____

Prof. M.Sc. Paulo Martins (UNESC)
Orientador

_____

Prof. M.Sc. Alfredo Engelmann Filho (UNESC)

_____

Prof. M.Sc. Ricardo Portes (Prefeitura Municipal de Florianópolis)

Dedico este trabalho a minha namorada,
família e a todos meus amigos.

AGRADECIMENTOS

Agradeço a todos aqueles que me apoiaram nesta caminhada mesmo estando tão ausente nos últimos tempos. Aos professores Paulo Martins e Alfredo Engelmann pela ajuda, apoio e idéias. Ao Fernando Fraga Rodrigues que me mostrou uma visão diferente da informática. Ainda ao Everton Michels e em especial a minha namorada Jaqueline Fogaça.

RESUMO

Com o aumento contínuo de crimes praticados por meio do computador, os chamados crimes digitais, a importância da evidência digital vem crescendo na mesma proporção e atualmente é reconhecida por profissionais da lei. A manipulação e o tratamento corretos das evidências digitais são primordiais para que as mesmas sejam aceitas pela comunidade científica relevante e perante a lei. Neste trabalho, serão apresentadas algumas formas publicamente conhecidas na análise das evidências em ambientes GNU/Linux. Analisando basicamente a memória, módulos do *kernel* e sistema de arquivos de maneira prática, a pesquisa não tem como objetivo esgotar o assunto e faz uso do embasamento teórico presente na literatura e a utilização de ferramentas livres.

PALAVRAS-CHAVE: Segurança; Crimes Virtuais; Perícia Forense; Servidores GNU/Linux.

ABSTRACT

With the continuous growth of computer based crimes, named digital crimes, the importance of the digital evidence is increasing at the same rate and has been accepted by professionals of the law. The right handling and processing of the digital evidences is primordial for it acceptance by the relevant scientific community and by the law. Will be presented in this paper some known public methods for the evidences analysis in GNU/Linux environments. Basically analyzing the main memory, kernel modules and file systems in practical manner, the research doesn't have as an objective to exhaust the subject and makes use of the current theoretical knowledge in existing literature and the use of free tools.

KEYWORDS: Security; Cybercrime; Computer Forensic; GNU/Linux Servers.

SUMÁRIO

1 INTRODUÇÃO

Com a proliferação dos computadores e o acesso cada vez mais fácil à Internet nosso comportamento e atividades do cotidiano vem sofrendo algumas mudanças, como a maneira como fazemos negócios ou lazer. Estas facilidades trouxeram muitos benefícios ao nosso dia-dia, porém toda esta tecnologia e facilidade que deveriam ser somente aplicadas para o nosso bem, também podem ser utilizadas para fins ilícitos, praticados por criminosos ao redor do mundo que pela falsa segurança de impunidade vem crescendo continuamente.

A criminalidade praticada por meios computacionais é um reflexo do mundo físico que acabou se adaptando à nova realidade. Dentre eles podem ser citados: fraude financeira, sabotagem dos dados e/ou rede, roubo de informações proprietárias, penetração em sistemas ou negação de serviço, acesso não autorizado por intrusos, abuso de acesso privilegiado da internet por funcionários e vírus.

Independentemente do propósito do criminoso, o mesmo sempre deixará para trás algumas “pistas” sobre suas atividades. Estas “pistas“ são chamadas de evidências, que devidamente capturadas e analisadas podem ser utilizadas como prova em um processo judicial, por exemplo. Para tanto é necessária uma investigação do(s) computador(es) envolvido(s) por especialistas que acabam se beneficiando das evidências residentes nos computadores, seja em um simples furto ou até mesmo espionagem industrial.

O processo de extração de dados relevantes que valham como prova, é chamada de perícia forense computacional. Executada por especialistas na busca e análise de evidências que sejam exatas e autênticas.

Seja qual for o propósito da perícia forense: auxílio em processos jurídicos, enquadramento em delito praticado na legislação nacional vigente ou ainda detecção de falhas no sistema. Alguns dos maiores desafios encontrados pelos peritos forenses são a falta de rotinas específicas para cada um dos inúmeros sistemas computacionais existentes e suas particularidades. Principalmente na análise dos vários dados gerados na aquisição das evidências, que vêm de diversas fontes (sistema de arquivos, memória principal, arquivos de *log*, estados do sistema operacional, entre outros) objetivando a reconstrução dos eventos do criminoso.

A etapa de análise das evidências é de suma importância, já que de nada adiantaria a aquisição dos dados gerados na busca de evidências, se o perito não interpretar os eventos do intruso ou até mesmo correlacionar às evidências na busca do(s) praticante (s) do delito.

1.1 OBJETIVO GERAL

Aplicar técnicas computacionais forenses na análise dos resultados gerados pela etapa de busca de evidências na memória principal, memória secundária (*Hard Disk*), processos e módulos do kernel em servidores GNU/Linux.

1.2 OBJETIVOS ESPECÍFICOS

Abaixo, uma lista de objetivos específicos a serem concluídos durante o desenvolvimento da pesquisa para que o objetivo geral seja alcançado:

a) abordar os principais aspectos de segurança em servidores GNU/Linux que serão relevantes ao enfoque da pesquisa;

b) compreender os principais passos na perícia forense computacional em servidores GNU/Linux;

c) discutir os vários aspectos que envolvem a análise de evidências na perícia forense em servidores GNU/Linux de maneira geral;

d) enumerar possíveis evidências da memória principal, memória secundária (*Hard Disk*), processos e módulos do kernel em servidores GNU/Linux;

e) realizar estudo de caso aplicando as técnicas computacionais forense estudadas na memória principal, memória secundária (*Hard Disk*), processos e módulos do kernel do GNU/Linux.

1.3 JUSTIFICATIVA

Alguns crimes ainda são resolvidos usando evidências como: impressões digitais, pegadas, documentos de papel e outros itens tangíveis extraídos da cena do crime. Com a popularização da tecnologia, a evidência digital também pode ser utilizada para este fim. O que muitas vezes pode ser mais proveitosa que uma impressão digital (SCHWEITZER, 2003).

Ainda segundo Schweitzer, os profissionais da lei estão reconhecendo evidências providas do computador e que podem ser incluídas como ponto chave na solução de certos crimes. Com o aumento da relevância da evidência digital cresce ainda mais a importância de que a evidência seja corretamente tratada e examinada.

Existe uma carência de bibliografias dedicadas à perícia forense no país. Principalmente direcionadas para objetivos específicos, abordando soluções práticas e mais aprofundados na análise das evidências em sistemas GNU/Linux. Observando que nas bibliografias pesquisadas o estudo fica em torno da busca de evidências e não na análise destas.

Tendo como base a defasagem de bibliografias e objetos de estudos se comparada a outras áreas da computação, como também pelo estudo do tratamento correto das evidências em áreas particulares dos sistemas GNU/Linux, este trabalho vem a contribuir com a perícia forense no que se refere à análise das evidências na memória principal, memória secundária (*Hard Disk*), processos e módulos do kernel em servidores GNU/Linux na perícia forense computacional.

1.4 ESTRUTURA DO TRABALHO

A presente pesquisa tem como meta demonstrar de maneira prática a análise das evidências em um ambiente GNU/Linux. Para tanto, o trabalho é divido basicamente em duas partes: a primeira contemplando a base teórica dos objetos de estudo e a segunda a parte prática, que é baseada fortemente na fundamentação teórica apresentada, simulando um ambiente real.

O primeiro capítulo apresenta o propósito do trabalho, que é seguido pela apresentação da perícia forense e os assuntos pertinentes que o envolvem. O terceiro capítulo tem como objetivo fundamentar toda a teoria envolvida em um ambiente Linux e seus componentes. Os dois próximos capítulos apresentam a segurança do Linux (administrativo e do sistema operacional), como também os objetivos e técnicas utilizadas por intrusos na intrusão, respectivamente. O sexto capítulo demonstra os principais componentes do Linux e o que eles podem oferecer ao perito na busca e análise das evidências. E completando o trabalho a parte prática, que utiliza uma simulação de um servidor para a prática da fundamentação teórica apresentada no decorrer da pesquisa.

2 PERÍCIA FORENSE COMPUTACIONAL

Para entender o significado correto de Perícia Forense Computacional, antes é preciso explicar algumas definições que abrangem esta área. A aplicação de estudos científicos a lei é chamada de ciência forense (CASEY, 2004). Ou seja, a utilização da ciência ou da tecnologia em processos investigativos, estabelecendo fatos ou evidências no tribunal de justiça (CARRIER, 2005). E a utilização da ciência forense em sistemas computacionais em investigações oficiais pelo perito judicial na elaboração de um laudo, é comumente chamada de Perícia Forense Computacional.

Perícia Forense Computacional é a coleta, preservação, análise e apresentação das evidências (VACCA, 2002). Que utiliza ferramentas e técnicas computacionais no ambiente investigado, auxiliando os juízes nas tomadas de decisões num processo judicial. Suprindo o mesmo com dados, informações e provas periciais que somente um perito ou *expert* no assunto pode abordar de forma com que sejam validadas judicialmente, adquiridas e analisadas sem perda ou contaminação das evidências.

A seguir serão demonstrados alguns temas relevantes que englobam esta área, no qual o profissional forense deve dominar para o bom desempenho do seu trabalho.

2.1 EVIDÊNCIAS DIGITAIS

Nos sistemas computacionais, não importa o quanto cuidadosos sejam os criminosos e o que eles façam para destruir provas incriminadoras, sempre acabarão deixando para trás sinais vitais que podem ser utilizados para traçar suas atividades e incriminá-los (VACCA, 2002). Estas “pistas” podem servir como prova ou álibi em um crime, que são chamadas de evidências digitais ou também evidências computacionais.

Evidência digital é definida como qualquer dado armazenado ou transmitido usando um computador que pode provar ou negar a teoria de como ocorreu um crime, intenção ou álibi (CASEY, 2004). Estabelecendo uma conexão entre um crime e a vítima ou um crime e um criminoso.

Os computadores são uma fonte preciosa de evidências e acabam “abrigando mais informações pessoais e segredos que os que poderiam ser encontrados em uma lixeira de 100 litros” (MANDIA, PROSISE: 2002). A remoção completa destas informações armazenadas é difícil, sejam elas excluídas intencionalmente ou por acidente (VACCA, 2002). Por exemplo, em sistemas de arquivos Ext2 o arquivo mesmo quando removido, os dados ainda continuam armazenados (desde que não sejam sobrescritos) sendo facilmente recuperados (CARRIER, 2005).

Como qualquer outro tipo de evidência, a digital deve ser: autêntica, exata, completa, convincente ao corpo de jurados, estar em conformidade com as leis e legislação (VACCA, 2002). Para tanto a manipulação destas deve ser feita por um perito forense computacional treinado, caso contrário ela pode ser

rejeitada pela justiça, já que é uma forma delicada de evidência e pode ser muito difícil de tratar. Ao mesmo tempo pode ser facilmente contaminada tornando mais difícil sua manipulação. Pode ainda ser alterada sem sinais óbvios de distorção por criminosos ou até mesmo no tratamento incorreto pelo perito (CASEY, 2004). Segundo Casey (2004), existem algumas particularidades quanto à manipulação destes tipos de evidências que ajudam os peritos, como:

a) pode ser exatamente duplicada e a cópia pode ser examinada como se fosse a original;

b) pode ser comparada com a original e verificada se houveram alterações;

c) é difícil de ser destruída realmente. Até mesmo quando o arquivo é "excluído" ou o *Hard Disk* é formatado, a evidência digital pode ainda ser recuperada;

d) ao se destruir evidências digitais, cópias de outras associadas a esta podem residir em outros locais.

2.2 PROVAS PERICIAIS

De maneira conceitual e generalizada, a prova é uma forma direta de conhecer a verdade de um fato. Através da aquisição e demonstração de documentos, declarações e outros que foram examinados com o objetivo de auxiliar e transpor a verdade dos fatos (JESUS, 2000).

As provas periciais podem incriminar ou não um indivíduo, portanto a declaração científica do caso deve ser somente emitida por pessoa de

conhecimento técnico especializado, fornecendo declarações científicas sobre fatos relevantes relacionadas à causa da ocorrência do evento.

Cada caso ou processo judicial pode abranger áreas diferentes, portanto serão exigidos profissionais especializados em cada atividade. Estes profissionais são chamados de perito ou *expert*, cuja área de atuação deve ser bem definida para a obtenção de evidências e provas da forma mais correta e precisa possível. Tais especialidades e profissionais da área serão abordadas a seguir.

2.3 PROFISSIONAIS E ESPECIFICAÇÕES DE ÁREAS

A divisão em áreas macros na perícia forense computacional, como: aquisição, processamento e análise das evidências digitais acabam exigindo especializações, habilidades e procedimentos diferentes (CASEY, 2004). Por exemplo: o profissional que coleta os dados de um disco precisa dominar a forma com que os dados residem nele e de que forma não “contaminá-los” na aquisição das evidências, mas para o profissional que analisa os dados coletados, não necessariamente precisa saber como funciona o armazenado dos arquivos, e sim o funcionamento do sistema de arquivos para através destes tentar recriar as atividades do criminoso.

Ainda segundo Casey (2004), os profissionais da área podem ser divididos de acordo com suas habilidades e especializações, como:

a) técnico da cena de crimes digitais: pessoa responsável pela aquisição dos dados;

b) examinador de evidências digitais: responsável pela produção detalhada de qualidade de evidências digitais;

c) investigador digital: responsável pela investigação completa. Também são responsáveis pela reconstrução de ações relativas ao crime usando as informações adquiridas pelas primeiras respostas e examinadores.

As ações desenvolvidas pelo perito forense exigem procedimentos específicos, mas que podem ser executadas de forma diferente levando em conta a grande diversidade de sistemas e aparatos tecnológicos envolvidos. Na tentativa de diminuir esta diferença, foram criadas algumas metodologias pelas comunidades científicas e instituições, dando maior credibilidade e solidez à perícia forense em casos judiciais.

2.4 METODOLOGIAS NO PROCESSO DE INVESTIGAÇÃO

O processo de investigação tem como objetivo principal descobrir e mostrar a verdade. Para tanto o foco é determinar o que aconteceu, onde, quando, como, quem estava envolvido e por que, aplicando protocolos completos, métodos comprovados e ferramentas confiáveis (CASEY, 2004).

Para que as investigações sejam mais confiáveis, algumas metodologias foram criadas através de discussões e experiências nesta área. A seguir será mostrada uma delas chamada Procedimentos Funcionais e Padrões ou SOP (*Standard Operating Procedures*) que serve principalmente como um guia

(*framework*) neste processo já que cada investigação pode ser considerada como única, com suas particularidades.

2.4.1 A Metodologia SOP

Incorpora princípios e técnicas da ciência forense, incluindo comparação, classificação, individualização e avaliação da fonte. Abaixo as etapas destes processos, que servem para a revelação da verdade (baseados sob prova) que ajudem na decisão do veredicto final:

2.4.1.1 Autorização e Preparação

Primeiramente antes de qualquer investigação, o perito forense deve se certificar de que não está infringindo nenhuma lei. Caso contrário sua perícia será enfraquecida ou simplesmente invalidada. Obrigatoriamente o perito precisa de autorização e o exame deve ficar restrito somente ao que foi determinado pelo juiz de direito.

2.4.1.2 Identificação

Levantamento das informações relevantes ao crime e identificação de todo o *hardware* do computador. Os tipos de informações a serem levantados

dependem do tipo de crime cometido e a habilidade de procurar no lugar certo dependem obviamente da experiência do perito forense.

2.4.1.3 Documentação

A documentação é essencial em todas as etapas da perícia forense. Primeiramente, porque caso algum outro perito tenha que dar seqüência na perícia e também porque a perícia terá maior credibilidade se a documentação estiver completa, com dados como:

a) quem coletou e tratou as evidências, com data e hora;

b) utilização de *hash* em todas as evidências copiadas para demonstrar que as cópias estão livres de alteração e são autênticas.

2.4.1.4 Coleta e Preservação

Após a identificação das fontes de evidências estas devem ser coletadas e mais tarde autenticadas. É primordial que as evidências não sejam alteradas, seja na coleta ou na preservação das mesmas.

Antes de copiar as evidências é aconselhável calcular o valor *hash* do disco original. Após a cópia deve ser feito a verificação para saber se é idêntica a original. É interessante fazer pelo menos duas cópias dos dados em discos “limpos” e de preferência com ferramentas diferentes (CASEY, 2004).

2.4.1.5 Exame e Análise

Fase posterior à coleta das evidências, onde estas serão examinadas e analisadas pelo perito na busca de provas.

2.4.1.6 Reconstrução

Busca do retrato completo de um crime, tenta responder as seguintes perguntas: o que aconteceu? Quem? Quando? Onde? Como? E Por quê?

Da metodologia apresentada, o presente trabalho abordará basicamente o exame e análise das evidências como também alguns aspectos relevantes à pesquisa como a coleta das evidências.

Mesmo se o perito estiver seguindo rigidamente metodologias internacionais de perícia como a mencionada, o perito forense deve sempre levar em consideração as leis e regras que regem o ambiente onde ele esta trabalhando. Por exemplo, estar de acordo com as regras internas de empresas (se for o caso), leis municipais, estaduais e federais para que a perícia não seja invalidada.

2.5 ASPECTOS LEGAIS

Em um processo judicial a lei brasileira coloca o juiz no topo da hierarquia, com a função de fazer valer a justiça. Para isto os juizes resgatam todas as informações que acharem necessários para chegar num livre convencimento, seja através da análise de documentos, provas ou interrogação das partes (JESUS, 2000). Caso o processo exija conhecimentos aprofundados o juiz tem o direito de recorrer a peritos para auxiliá-lo na busca da verdade dos fatos que são inteligíveis aos leigos. Desta forma, os juízes têm como fundamentar cientificamente suas decisões.

O juiz deverá formular as questões que achar necessária para esclarecimento do caso ao perito. Que por sua vez poderá utilizar de todos os meios necessários: ouvir testemunhas, obter informações, solicitar documentos, instruir laudos com plantas, desenhos, fotografias e outras peças quaisquer. Caso seja necessário, o juiz poderá pedir uma nova perícia quando a primeira não for suficientemente esclarecedora. Esta por sua vez deve possuir o mesmo objetivo da primeira, mas com o propósito de corrigir as eventuais omissões ou erros da anterior. Lembrando que a segunda perícia não substitui a primeira, cabendo ao juiz apreciar livremente o valor de uma ou de outra. Mesmo utilizando os trabalhos periciais, os juizes são livres em suas convicções sobre os fatos, não estando restritos à prova pericial, podendo, se assim entenderem, desprezá-la (JESUS, 2000).

Os peritos são instituídos como profissionais de nível universitário, devidamente inscritos no órgão de classe competente e que neste tenha

especialidade comprovada. Dando ao juiz a escolha de livre arbítrio, em caso de na localidade em questão não ter profissionais que preencham estes requisitos. Como também, substituir o perito se este carecer de conhecimentos técnicos ou não cumprir seu trabalho sem motivo legal.

A perícia tem como objetivo a elaboração do laudo pericial, onde serão descritos minuciosamente os exames efetuados, que devem responder os questionamentos feitos pelo juiz de direito. O perito que faltar com a verdade por má fé responderá pelos prejuízos que causar as partes e ficará inabilitado de praticar outras pericias num prazo de 2 (dois) anos. Podendo ser condenado por falsa afirmação, negar ou calar a verdade a um (1) ou 3 (três) anos de reclusão e multa, estabelecido pelo Código Penal.

Um dos pontos mais importantes, focalizando agora a perícia forense computacional são as provas. A prova pericial é regulada pelo código, sendo considerada de elevada importância por todos que lidam no mundo jurídico (JESUS, 2000). Em sistemas computacionais podem ser adquiridos através de documentos eletrônicos, como: texto escrito, desenho, fotografia digitalizada, sons, vídeos, ou seja, tudo que puder representar um acontecimento e que esteja armazenado em um arquivo digital.

O direito é um conjunto de regras ou normas sociais obrigatórias que tentam assegurar o equilíbrio da sociedade, podendo ser dividida em Civil ou Penal. A primeira estatui normas relativas ao estado e capacidade das pessoas, ou seja, direito individual. A segunda consiste o meio de prevenção, repressão e punição dos fatos considerados atentatórios à ordem social. Desta forma, a área do direito que rege a conduta criminosa enfocada nesta pesquisa é a penal.

A legislação penal vigente não contempla leis específicas que abordem de maneira clara o assunto crimes digitais. Desta forma, em um processo judicial cabe a interpretação entre os crimes praticados através de sistemas computacionais e os praticados através do meio físico. Ou seja, podem ser resolvidos ou dirimidos com a simples aplicação dos institutos tradicionais do direito, sendo somente necessária a correta interpretação da legislação vigente. O desvio de dinheiro de uma conta bancária via Internet é um furto como qualquer outro, por exemplo, diferenciando apenas pelo autor e o meio utilizado.

Abaixo, segue alguns exemplos de interpretações que podem ser utilizadas em um processo judicial extraídos do Código Penal na busca de punição e crimes praticados com uso de sistemas computacionais:

a) penetração de sistemas e/ou rede, negação de serviço, criação ou propagação de vírus, pichação de sites: pode ser usado o Art. 163, que consta a destruição, inutilização ou deteriorização de coisa alheia;

b) roubo e desvio de dinheiro de contas bancárias: furto (Art. 155), roubo (Art. 157) e apropriação indébita (Art. 168). Dependendo do caso cabe o Art. 288 quanto a formação de quadrilha e receptação (Art. 180) na utilização de "laranjas";

c) outros: extorsão (Art. 158), ameaça (Art. 147), divulgação de segredo (Art. 153 e 154), Incitação ao Crime (Art. 286), calúnia e injúria (Art. 138 e 140) e pornografia e pedofilia (Art. 234).

Como já mencionado, a perícia forense tem como objetivo auxiliar diversas áreas onde são exigidos conhecimentos técnicos aprofundados como em

casos judiciais. Com o aumento contínuo de crimes digitais envolvendo meios eletrônicos, as evidências digitais estão cada vez mais sendo aceitas e a manipulação adequada das evidências é essencial, já que pode distorcer a realidade dos fatos e interferir no andamento de um processo, podendo até mesmo levar um inocente para a cadeia ou um infrator à liberdade. Por conseqüência disso, para que a perícia seja aceita, são exigidos metodologias e procedimentos adequados que estejam de acordo com as leis vigentes e ao mesmo tempo sejam aceitos pela comunidade científica relevante.

Além de estar envolvido sob os aspectos legais, é essencial que o perito forense tenha os mínimos conhecimentos técnicos necessários, aplicáveis ao ambiente em que a investigação será realizada. A base dos conhecimentos do perito deve ter um embasamento teórico científico forte, para que seu parecer seja relevante e aceito sem que haja muitas contestações.

Nos próximos capítulos, será apresentado o embasamento teórico mínimo necessário para um processo investigativo em servidores GNU/Linux.

3 FUNDAMENTOS DOS SERVIDORES GNU/LINUX

Antes de qualquer coisa, para o perito forense realizar um bom trabalho ele precisa entender e dominar os principais conceitos e funcionalidades dos vários aspectos computacionais que podem envolver sua perícia. Ao periciar um servidor GNU/Linux, por exemplo, o perito terá que manipular seus dados de forma a não alterá-los, detectar e restaurar arquivos do disco, analisar processos e dados da memória, avaliar serviços, descobrir artimanhas e armadilhas deixadas pelo intruso, dentre outras funções. Para tanto, o perito deve possuir conhecimentos concretos sobre o funcionamento e como se comporta o sistema analisado. Abaixo serão tratados alguns aspectos pertinentes, focados apenas para o propósito da pesquisa em questão, no qual o perito forense deve ter domínio para uma boa perícia.

3.1 O QUE É LINUX?

O Linux é um sistema operacional moderno, membro da família Unix e que vem se popularizando a partir do inicio dos anos 90 (BOVET, CESATI: 2003). Por ter surgido das entranhas do Unix que já possuía mais de 20 anos de amadurecimento, o Linux é a sua melhor variante, rápido, robusto, completo e utiliza os melhores recursos do Unix e eliminando os piores (MAXWELL, 2000). É compatível com o padrão POSIX e possui seu código fonte sob a licença GNU. O que significa que seu código fonte é portável para qualquer outro sistema

operacional também compatível com POSIX e também é aberto, podendo ser estudado e modificado livremente.

O Linux, por si só, é o *Kernel*. Que pode ser considerado o coração, a mente e o sistema nervoso deste sistema operacional (MAXWELL, 2000). O *Kernel* sozinho não tem funcionalidade alguma, que só num todo se torna útil (ZILLI, 2003). Esta junção entre o Kernel e as mais diversas ferramentas é comumente chamada de GNU/Linux.

Além de trabalhar com inúmeras arquiteturas, como: Intel, Alpha, SPARC, Motorola MC680x0, PowerPC e IBM System/390. Segundo Bovet e Cesati (2003), o Linux possui as principais características:

a) *kernel* do Linux é monolítico;

b) *kernel* pode carregar ou descarregar dinamicamente algumas partes do código que são chamados de módulos;

c) o k*ernel* do Linux usa *thread* em funções periódicas;

d) possui suporte a aplicações *multi-thread*;

e) o k*ernel* do Linux não é preemptivo.

f) possui suporte a multiprocessadores, Suporte a *Symmetric Multiprocessing* (SMP);

g) possui suporte a diversos sistemas de arquivos.

Ainda segundo Bovet e Cesati (2003), o Linux oferece as seguintes vantagens sobre os concorrentes comerciais:

a) linux é livre;

b) linux é inteiramente customizável;

c) linux roda em hardware barato;

d) linux possui um padrão alto na qualidade do código fonte;

e) linux pode ser muito pequeno e compacto;

f) linux é bem suportado.

Sob o aspecto da perícia forense, este sistema operacional possui vantagens já que suporta muitos sistemas de arquivos diferentes, possui ferramentas que auxiliam nesta tarefa, além de ser possível utilizar a saída de uma ferramenta como entrada em outra, dando uma gama grande de possibilidades para o perito (CASEY, 2004), (MANDIA, PROSISE:2002). É possível utilizar o Linux na coleta e análise de evidências em vários sistemas operacionais, como em ambiente Windows, por exemplo. Contando com a vantagem de poder estudar e aprender seu funcionamento, podendo ainda melhorá-lo, incluindo ou aprimorando suas funcionalidades para perícias em ambientes específicos.

3.2 PROCESSOS

Processo é uma abstração que representa uma instância de um programa em execução (OLIVEIRA, CARISSIMI, TOSCANI: 2004). Por exemplo, ao executar um editor de texto é gerado um processo, ao executar um segundo editor outro processo é iniciado e que utilizam o mesmo código, mas são processos distintos no qual o kernel trata de maneira independente. Todo processo possui uma identificação única no qual o kernel utilizará como referência, que é um número inteiro entre 0 e 32.767. Esta identificação é chamada de *process ID*

ou simplesmente PID, no qual somente processos especiais para o sistema recebem os PIDs 0 ou 1. O restante dos processos “normais” recebem os demais números disponíveis (MAXWELL, 2000). Os processos ainda podem ser divididos em processos de usuário e de sistema (*daemons*).

Por padrão, o kernel de sistemas baseados em Unix diferencia o modo de execução do processo em modo usuário (*user space*) e modo *kernel* (*kernel space*), de acordo com o estado da CPU. Um processo sendo executado em modo usuário não possui acesso direto a estrutura de dados ou programas do *kernel*, já que processos que rodam no nível do *kernel* estas restrições não se aplicam (BOVET, CESATI: 2003). Para acessar determinadas funcionalidades disponíveis apenas ao modo *kernel*, um processo precisa invocar funções especiais denominadas chamadas de sistema (*system calls*) que a CPU processa em modo *kernel*, retorna ao modo usuário e devolve o resultado ao processo solicitante.

Cada processo é executado em um endereço limitado de memória, este espaço que o processo enxerga, acessa e pode executar suas instruções é chamada de memória lógica. O espaço de endereçamento lógico de um processo é formado por todos os endereços lógicos que esse processo pode gerar e cada processo possui seu próprio espaço de endereçamento lógico. Para fazer o mapeamento lógico aos físicos (circuitos integrados de memória) o Linux utiliza a Unidade de Gerenciamento de Memória (*Memory Management Unit*) ou MMU.

Uma forma de comunicação entre os processos são os sinais, que é por onde o sistema notifica o processo de um evento qualquer, que podem ser de dois tipos: notificação ou erro e exceções. O processo ao receber o sinal pode tratá-lo ou ignorá-lo. Caso o processo não trate o sinal, o kernel assume, podendo:

a) ignorar o sinal;

b) suspender o processo;

c) retomar o processo caso estivesse parado;

d) terminar o processo;

e) escrever o contexto e conteúdo da execução da memória em um arquivo (*core dump*) e terminar o processo.

3.3 MÓDULOS DO KERNEL (LINUX KERNEL MODULES)

Tradicionalmente, os núcleos (*kernel*) de sistemas operacionais são organizados de duas formas diferentes: monolítico e *microkernel*. O *kernel* monolítico possui todos seus componentes do sistema operacional em um código único, enquanto o *microkernel* possui apenas um conjunto reduzido de funcionalidades. O Linux é um sistema monolítico, mas que implementa a vantagem da modularidade, portabilidade e uso otimizado de memória através da utilização de módulos. Dessa forma, o Linux possui o desempenho de um *kernel* monolítico e as vantagens do *microkernel* (OLIVEIRA, CARISSIMI, TOSCANI: 2004).

Um módulo é um arquivo objeto cujo código pode ser ligado ou desligado ao *kernel* dinamicamente. Ao ser adicionado, o código do módulo passa a fazer parte do código do sistema operacional, como se fosse uma outra funcionalidade estática do *kernel*, garantindo seu processo em modo *kernel* (BOVET, CESATI: 2003). Estes módulos possuem diversas funcionalidades

como, por exemplo, sistema de arquivos, *drivers* de dispositivos e são carregados ou descarregados em tempo de execução através de chamadas de sistemas que o Linux dispõe.

3.4 MEMÓRIA

Quando um processo é iniciado, o *kernel* precisa disponibilizar espaço na memória onde ficará armazenado o código executável do programa, variáveis, entre outros dados. O gerenciamento de memória é feito pelo *kernel*, que é sua tarefa mais complexa (BOVET, CESATI: 2003). A memória RAM ou principal é dividida em duas partes, uma contém a imagem do *kernel* (código e estrutura de dados estáticos) e o restante é utilizado para a memória virtual do sistema.

A memória virtual é uma camada abstrata entre o processo e o hardware. O Linux utiliza este modelo de abstração para tornar transparente ao processo o acesso à memória independentemente de onde estejam os dados, seja na memória RAM ou disco. Por exemplo, o sistema pode querer utilizar mais memória que a RAM disponível, então o Linux utiliza um espaço reservado em disco (*swap*) para este propósito, esta técnica é denominada *swapping* ou troca.

A memória RAM também pode ser utilizada como *cache* para o sistema de arquivos e outros dispositivos. Dessa forma, o sistema eleva seu desempenho já que o acesso ao disco é 4 (quatro) ou 5 (cinco) ordens de grandeza (de 10000 a 100000 vezes) mais lento que um acesso à memória principal (OLIVEIRA, CARISSIMI, TOSCANI: 2004).

3.5 SISTEMA DE ARQUIVOS EXT2

Sistemas de arquivos é a forma utilizada pelo sistema operacional para armazenar e encontrar dados. Este mecanismo ajuda usuários a manipular seus dados através de arquivos e diretórios de forma hierárquica (CARRIER, 2005). O *hardware* simplesmente disponibiliza espaço em disco na forma de setores, e o sistema de arquivos implementa recursos de software que o *hardware* não disponibiliza (OLIVEIRA, CARISSIMI, TOSCANI: 2004).

O sistema de arquivos nativo no Linux é o Ext2, que possui boas características de sistema de arquivos modernos com alto desempenho. Ext2 é utilizado na maioria dos sistemas Linux (BOVET, CESATI: 2003), apesar do Linux suportar diversos sistemas de arquivos, como: ext, ext2, ext3, xia, minix, umsdos, msdos, vfat, proc, smb, ncp, iso9660, sysv, hpfs, affs e ufs.

O Linux organiza seu sistema de arquivos em uma árvore hierarquizada, resultando uma estrutura única que agrega todas as informações relativas ao sistema de arquivos (OLIVEIRA, CARISSIMI, TOSCANI: 2004).

Segundo Carrier (2005), o sistema de arquivos Ext2 para melhor analisá-lo e estudá-lo pode ser divididas em quatro categorias, denominadas: sistema de arquivos, conteúdo, meta dados e nome de arquivo.

3.5.1 Sistema de Arquivos

A categoria de sistema de arquivos é onde são definidas as configurações e o formato do Ext2. Nela estão contidas as estruturas de super bloco e descritores de grupos de blocos (CARRIER, 2005).

3.5.1.1 Super Bloco

As informações básicas do sistema de arquivos ficam armazenadas nesta estrutura de dados, tais como: tamanho do bloco, número total de blocos, número de blocos por grupo de blocos, número de blocos reservados antes do primeiro grupo do bloco, número total de *inodes* e de *inodes* por grupo de bloco. Além disso, o super bloco contém: número total de *inodes* e blocos livres, usados quando seja necessário alocar *inodes* e blocos, como também o nome do volume, tempo da última escrita, última montagem e o caminho onde o sistema de arquivos foi montado.

3.5.1.2 Tabela Descritora de Grupos dos Blocos

Cada grupo de blocos do sistema de arquivos contém uma tabela no qual contém dados administrativos para o grupo em questão, descrevendo onde alguns dados podem ser encontrados (apontamentos), tais como: bloco de mapa de bits de blocos, bloco de mapa de bits de *inodes*, primeiro bloco da tabela de

inodes. Além disso, possui contador para o número de blocos e *inodes* livres e o número de *inodes* alocados para os diretórios.

O Ext2 utiliza mapa de bits para gerenciar a alocação e liberação de blocos e *inodes*. O mapa de bits de blocos refere-se aos blocos que abrangem do primeiro bloco ao último bloco do grupo. Da mesma forma, o mapa de bits de *inodes* refere-se aos *inodes* que abrangem o primeiro *inode* do grupo ao último *inode* do grupo.

3.5.2 Conteúdo

O Ext2 usa blocos como unidade de dados, que pode ser de 1,024; 2,048; ou 4,096 bytes no qual é especificado pelo super bloco. Um bloco é um grupo de setores consecutivos que engloba o conteúdo de diretórios e arquivos. Todos os blocos possuem um endereço, iniciando com 0, e o bloco 0 é localizado no início do setor do sistema de arquivos. Como já mencionado, a alocação dos blocos é gerenciada pelo mapa de bits de blocos, onde cada bit corresponde a um bloco dentro do grupo.

3.5.3 Meta dado

São os dados que descrevem os arquivos e diretórios, como por exemplo: atributos, permissões, tamanho, localização dos blocos de dados, entre outros.

3.5.3.1 Inodes

Cada arquivo ou diretório do sistema de arquivos são representados pelo *inode,* que são alocados para armazenar suas informações e são definidos no super bloco e armazenados na tabela de *inodes* que é apontada pelo descritor do grupo. O seu gerenciamento de alocação e liberação dos *inodes* fica por conta do mapa de bits de *inodes*.

A informação do dono é armazenada usando o ID do grupo e do usuário. Para informações temporais, o *inode* contém tempo do último acesso, modificação, mudança e exclusão, também chamados de *MAC Time*. O tempo é armazenado como o número de segundos desde o dia 1 de Janeiro de 1970 UTC.

Abaixo os campos utilizados para este gerenciamento:

a) *a-time*: corresponde o tempo em que o conteúdo do arquivo foi acessado;

b) *m-time*: corresponde a última vez que o conteúdo do arquivo foi modificado, e o último tempo de mudança;

c) *c-time*: corresponde a data em que o meta dado foi modificado. Quando um arquivo é criado, todos os três valores são atualizados para a hora corrente;

d) *d-time*: corresponde a exclusão do arquivo.

O *inode* pode identificar vários tipos de arquivos, como: arquivo regular, diretório, FIFO, dispositivos de bloco, dispositivos de caractere, *link* simbólico e *socket*. Onde cada um deles possui determinadas permissões para leitura, escrita e execução que podem sofrer determinadas mudanças de acordo com as propriedades especiais do *inode*, como por exemplo, "*sticky bits*", SUID e SGID.

3.5.3.2 Algoritmo de Alocação

Na alocação de *inodes* sejam eles arquivos ou diretórios, algumas regras para a melhor organização do sistema de arquivos são usadas, mas nem sempre podem ser seguidas (CARRIER, 2005):

a) O *inode* para o novo arquivo é alocado no mesmo grupo do *inode* do diretório pai;

b) *Inode* são alocados uniformemente entre grupos.

Abaixo o algoritmo simplificado de alocação de arquivos e diretórios no Ext2 segundo Carrier (2005):

O primeiro passo é descobrir em qual grupo de bloco o *inode* será alocado. Se for um arquivo, o Linux tenta alocar no mesmo grupo que o do pai. Desta forma, o Linux tenta alocar informações de um mesmo diretório na mesma área. Caso não possua *inodes* ou blocos livres no grupo, uma busca é feita em outros grupos.

No caso de ser um diretório, o objetivo é assegurar que os grupos sejam igualmente utilizados, desta forma, o Linux procura por um grupo que não esteja sendo muito utilizado. O primeiro passo desta busca é calcular a média de números de *inodes* livres e blocos livres por grupo. Este é calculado usando o número de *inodes* livres e blocos, que são armazenados no super bloco. O Linux então procura cada um dos grupos, iniciando com 0, e usa o primeiro que pertence a ele e que a média das soma de *inodes* e blocos são menores que a média. Se não encontrar ambos os valores que sejam menores que a média, os grupos com menor número de diretórios e maiores números de *inodes* livres é usado. Ambos os valores estão armazenados no descritor de grupos.

Quando alocado o conteúdo do *inode* é limpo e o *m-time*, *a-time* e *c-time* são “setados” para o horário atual. O *d-time* é “setado” em 0. O *link count*, que identifica quantos nomes de arquivos são apontados ao *inode* é “setado” em 1 para o nome do arquivo. Diretórios também possuem um *link count* de 2 por ter a entrada “.” do diretório.

Cada *inode* pode armazenar diretamente o endereço de apenas 12 blocos de seus dados. Se um arquivo precisar mais que 12 blocos, um bloco é alocado para armazenar os endereços dos restantes. Caso seja preciso mais blocos

que um endereço possa armazenar, mais blocos serão usados como dupla ou tripla de apontamentos indiretos.

Abaixo o algoritmo simplificado de exclusão de arquivos no Ext2 segundo Carrier (2005):

Cada *inode* contém um contador chamado de *link count*, que é igual ao número de arquivos que apontam para ele. Quando este valor é igual a 0 e nenhum processo tem o arquivo aberto, o mesmo é liberado. Quando o valor é igual a 0 e um processo tem o arquivo aberto, o Linux o torna órfão e é adicionado a uma lista no super bloco. Quando o processo fecha o arquivo, ou então o sistema é reiniciado, o *inode* é liberado.

Ext2 e ext3 manipulam a exclusão de arquivos de maneira diferente. Ext3 seta o tamanho do arquivo com 0 e limpa os ponteiros do bloco no *inode* e nos blocos indiretos. Ext2, por outro lado, não limpa estes valores, o que facilita a recuperação do arquivo.

3.5.4 Nome de Arquivo

No sistema de arquivos Ext2 um arquivo regular ou diretório pode ter nomes e localizações diferentes. Por exemplo, o arquivo /home/usuario/file.txt pode estar também localizado no /arquivo.txt, sendo que tem nome e localização diferentes, mas que apontam para o mesmo *inode* e conseqüentemente possuem o mesmo conteúdo. Para tanto o Ext2 utiliza o conceito de entrada de diretório e de link, que podem ser: *hard link* e *soft link*.

3.5.4.1 Entrada de Diretório

Uma entrada de diretório é uma simples estrutura de dados que contêm o nome do arquivo e o endereço do *inode* onde o meta dado do arquivo pode ser encontrado. Na sua estrutura de dados, a entrada de diretório armazena o número do *inode*, o nome, como também uma espécie de "apontador" para a próxima entrada de diretório válida. Este apontador é um número inteiro (*offset*), que adicionado ao endereço da entrada de diretório atual dá o endereço inicial da próxima entrada de diretório válida. Por questões de desempenho o nome da entrada do diretório deve ser múltiplo de 4, caso o nome não preencha o tamanho necessário o restante é valorado com caracteres null ("\0") (BOVET, CESATI: 2003).

Ao excluir uma entrada de diretório, o Linux "seta" o campo *inode* para 0 e soma o valor do *offset* desta entrada ao do anterior, desta forma a entrada excluída não será mais listada.

3.5.4.2 Links

Um *link* é um nome adicional para o arquivo ou diretório, habilitando chamá-lo pelo nome original ou pelo *link*. Como já mencionado, podem ser divididos em *hard* e *soft link*. Um *hard link* só pode ser usado no mesmo sistema de arquivos. Para ser criado, uma nova entrada no diretório é adicionada, que aponta para o *inode* do arquivo original, incrementado em 1 o *link count* do *inode*. Já um *soft link* pode ser usado em sistemas de arquivos diferentes. Para tanto, o

SO cria um *link* simbólico (tipo de arquivo especial), sendo armazenado em oito blocos alocados ao arquivo ou dentro do *inode* se o caminho é menor que 60 caracteres.

O Linux é um sistema operacional completo e possui características peculiares, algumas herdadas do Unix e outras próprias. A possibilidade de aprofundamento nos estudos de suas características e comportamento por causa do código fonte aberto, é apenas um dos motivos que fazem do Linux objeto de estudo e utilização em diversas áreas, incluindo a perícia forense. O aprofundamento neste sistema operacional torna-se necessário já que ele pode ser alvo de ataques, como também ser utilizado como ferramenta em ato pericial.

Somente após ter adquirido um conhecimento sólido do sistema a ser analisado, o perito forense terá condições de avaliar e entender os itens de segurança implementados pelo sistema.

4 ASPECTOS DE SEGURANÇA EM SERVIDORES GNU/LINUX

Por ser multi-tarefa, multi-usuário e utilizado em servidores, o Linux possui vários aspectos referentes a segurança que serão apresentados a seguir. Para melhor apresentação, será dividido em dois grupos: segurança quanto à administração e segurança inerente ao sistema operacional e aplicativos.

4.1 SEGURANÇA DE ADMINISTRAÇÃO

Os componentes da arquitetura de segurança do Linux podem ser especificados em: conta de usuários, controle de acesso arbitrário (DAC), controle de acesso de rede, criptografia, sistema de *logs* e detecção de intrusão (RAY, 2001). Abaixo um resumo de cada um deles:

4.1.1 Conta dos Usuários

O Linux é um sistema multiusuário, portanto pode haver vários usuários “logados” no sistema simultaneamente, que por sua vez pode estar conectado em mais de um terminal. Um dos controles mais básicos que o Linux impõe sobre estes usuários é a divisão em grupos, onde cada um deles possui características e funções diferentes no sistema.

4.1.1.1 Usuários Root ou Super Usuário

O *root* possui controle total do sistema. Pode manipular qualquer arquivo, executar e matar processos, gerenciar usuários, entre outros.

4.1.1.2 Usuários Comuns

Usuários que podem fazer *login* no sistema, mas que possuem acesso restrito aos arquivos e diretórios, não podendo executar muitas funções de nível do sistema.

4.1.1.3 Usuários do Sistema

Usuários que não podem fazer *login* no sistema, pois não possuem um *shell* e tem propósitos específicos de serviços do sistema, por exemplo: *nobody* para serviços HTTP, *mail* para serviços de *e-mail*.

Os usuários devem pertencer a grupos no qual pode ser definidas propriedades para o grupo e consequentemente herdado aos usuários integrantes.

4.1.2 Controle de Acesso Arbitrário – DAC (*Discretionary Access Control*)

Alguns arquivos, por exemplo, podem ser escritos apenas por um usuário, porém os demais têm somente permissão de leitura. No Linux esta funcionalidade se chama Controle de Acesso Arbitrário - *Discretionary Access*

Control (DAC), onde cada arquivo e diretório possuem permissões e atributos que especificam de qual forma o usuário pode interagir.

Outro aspecto importante em sistema de arquivos Ext2 são os atributos. O objetivo é aumentar a segurança de arquivos e diretórios. Por exemplo, se for utilizado o atributo “i”, o arquivo não pode ser alterado, excluído ou alterado, enquanto se o atributo for “s” o arquivo ao ser excluído os blocos são zerados.

4.1.3 Controle de Acesso de Rede

Linux também fornece controle de acesso de rede, ou seja, o Linux possui a habilidade de seletivamente permitir usuários e *hosts* conectarem-se uns aos outros.

4.1.4 Criptografia

Controle de acessos e gerenciamento de rede conta que com uma grande variedade de mecanismos de criptografia.

4.1.5 Sistema de *Logs*

Na arquitetura de segurança do Linux, o sistema de *log* é vital. São os *logs* que podem fornecer evidências reais de um ataque. Para isso fornece *logs* a nível de rede, *host* e de usuários, por exemplo:

a) *log* de todas as mensagens do *kernel* e sistema;

b) *log* de cada conexão de rede com IP de origem, e em alguns casos nome do usuário e sistema operacional do intruso;

c) *log* dos arquivos requisitados pelo usuário;

d) *log* de processos sob controle do usuário;

e) *log* para todos os comandos executados por um usuário específico.

4.1.6 Detecção de Intrusão

Detecção de intrusão é relativamente uma nova ciência e poucos sistemas operacionais vem equipado com estas ferramentas. Com estas podem-se ter algumas capacidades, como:

a) pode “logar” tentativas de intrusão e mostrar quando ocorrer;

b) ter ações pré-definidas quando um ataque for adequado a algum critério.

4.2 SEGURANÇA DO SISTEMA OPERACIONAL

Os aspectos de segurança apresentados abaixo são próprios do sistema operacional, ou seja, são na maioria transparentes para o usuário final.

4.2.1 Processos

O Linux associa para cada processo credenciais, no qual determinam o que cada processo pode ou não fazer. Ligando o processo a grupos ou usuário específicos (BOVET, CESATI: 2003). Por exemplo, ao acessar um arquivo o sistema verifica se o acesso do processo é legal, verificando as permissões (UID efetivo) do arquivo e as credenciais do processo.

Neste sistema de credenciais um processo para poder trocar o horário do sistema, por exemplo, teria que ter status de *root*, o que ocasionaria uma falta de segurança, já que se o processo tiver em poder de usuários maliciosos, estes poderão realizar qualquer ação no sistema. Para resolver este tipo de problema, o Linux também implementa o conceito de *capability* (capacidade). As *capabilities* permitem que um processo somente tenha acesso às tarefas específicas, sendo que para uma segurança maior, estes privilégios são perdidos assim que foram executados (MAXWELL, 2000). Por exemplo, se o processo precisa trocar o horário do sistema, ele teria somente acesso a esta permissão, não podendo, por exemplo, terminar outro processo. A principal vantagem das *capabilities* é que mesmo que o processo seja usado para fins maliciosos, este fica restrito a um

número limitado de operações e não a todas, como no sistema de credenciais (BOVET, CESATI: 2003).

Outra forma que o Linux utiliza para a segurança de processos são as chamadas de sistemas (*system calls*). Um processo comum sendo executado no espaço do usuário possui diversas restrições mesmo sendo executado com privilégios de *root*. Para abrir um arquivo, por exemplo, o processo precisa estar sendo executado no espaço do *kernel*. Dessa forma, o sistema operacional disponibiliza diversas chamadas de sistemas, que não passam de funções prontas, requeridas por interrupções executadas pelo processo e executadas pelo *kernel* que posteriormente são retornadas ao processo. Resumindo, chamadas de sistemas são uma espécie de camada extra de segurança, entre o processo e o *hardware*, que analisa a requisição antes de executá-la e protege os processos (BOVET, CESATI: 2003).

4.2.2 Memória

Áreas de código e variáveis de sistema não podem ser acessadas por usuários e processos indevidos, caso contrário o sistema pode ser corrompido. O Linux é capaz de proteger a memória de processos, reservando para cada processo seu endereçamento lógico de memória. Dessa forma, um processo ilegal não pode acessar áreas de memória não reservadas a ele.

Caso o processo tente acessar um endereço fora de sua área, é gerada uma interrupção. O Linux entra em modo do *kernel* e aborta o processo que tentou acesso ilegal à memória.

Como já mencionado, o Linux possui vários itens de segurança, seja no nível administrativo ou inerente ao sistema operacional. Mesmo com estes cuidados, os intrusos são capazes de burlar e explorar suas falhas. Entender e dominar a segurança do Linux é essencial para que o perito saiba como funciona seu comportamento, já que estes utilizam de todos os meios para ultrapassar esta barreira. Desta forma, o perito poderá entender como funciona o comportamento dos meliantes e saber onde procurar por evidências deixadas por eles. Este item será abordado no próximo capítulo da pesquisa.

5 MODO DE OPERAÇÃO

O comportamento do intruso na sua caminhada ao sistema e seus objetivos são peças chave no processo de investigação. Seu estudo possibilita o entendimento mais completo de onde podem residir possíveis evidências.

5.1 OBJETIVOS

A maioria dos usuários e administradores Linux considera que suas máquinas e servidores não são suficientemente importantes para serem invadidas. Estes é que acabam sendo as vítimas mais fáceis, seja como objetivo final do ataque ou apenas como uma “ponte” para um objetivo maior (MANDIA, PROSISE: 2002). Os objetivos das intrusões podem variar em muito, de acordo com conhecimentos e habilidades do infrator. Abaixo, uma lista dos objetivos mais básicos:

a) largura de banda: Os servidores podem ser utilizados como ponto de apoio a outros ataques, sendo utilizado como ferramenta a outra invasão para ocultar seus rastros ou ainda como parte integrante de um grupo em um ataque de negação de serviço distribuído (*Distributed Denial of Service* ou *DDoS*).

b) utilização de CPU: Infratores podem executar programas através da máquina invadida, não precisando utilizar suas próprias máquinas para isso. Por serem mais robustas que computadores pessoais, servidores de grande porte podem ser utilizados para quebrar senhas, por exemplo.

c) espaço em disco: Servidores invadidos podem ser utilizados como depósito dos mais diversos tipos de dados. Esta é uma das formas utilizadas na distribuição de softwares piratas.

d) roubar dados: Muitos dados valiosos podem ser adquiridos em servidores, incluindo: segredos industriais, dados bancários, cartões de créditos, como também dados pessoais e e-mails, por exemplo.

5.2 MODO DE OPERAÇÃO

Antes de serem levantadas as técnicas envolvidas no processo de ataque e de intrusão em servidores, se fazem necessário uma abordagem geral das etapas percorridas por um intruso até chegar ao seu objetivo final. Vale abordar ainda a distinção entre ataque e intrusão: ataques podem ser cometidos sem ganhar acesso à rede ou sistema, por exemplo: ataque DoS. Já uma intrusão ocorre quando o acesso foi realmente adquirido (SHINDER, TITTEL: 2002).

Segundo Cole (2002), existe um caminho típico, mas não padrão, usado pelos intrusos para ganhar acesso e explorar um sistema, destacando:

5.2.1 Coleta de Informações Passiva

Esta etapa é análoga a roubos de residências, onde os ladrões rondam as casas para saber todos os horários dos ocupantes, fazendo o levantamento de janelas, portas, e outras formas de acesso a ela. Do ponto de vista tecnológico, podem-se citar: coleta de informações em sites de busca ou no próprio site da vítima, busca de endereços de domínios, enumeração de redes, servidores, e-mails, entre outros (MANDIA, PROSISE:2002). Outra forma utilizada é a engenharia social, em que o intruso utiliza toda a sua esperteza para levantar mais dados, como por exemplo se passando por técnico terceirizado e coletar informações pelo telefone ou ficar “amigo” de um funcionário responsável pelo servidor.

5.2.2 Coleta de Informações Ativas

Após uma coleta de informações mais superficial, a coleta ativa é a forma de descobrir informações mais específicas, como:

a) servidores acessíveis;

b) localização de roteadores e firewalls;

c) identificação dos sistemas operacionais rodando;

d) portas abertas nos servidores e equipamentos;

e) serviços rodando nos servidores;

f) versões dos softwares.

Seguindo a analogia do roubo de residências, pode-se dizer que é a verificação de que as portas e janelas estão abertas, quais tipos de fechaduras usam e se existe sistema de alarmes.

5.2.3 Explorando o Sistema

5.2.3.1 Formas de Obter Acesso

a) ataques ao sistema operacional: as portas abertas e serviços rodando no servidor são os pontos de acessos. Quanto menos portas abertas e serviços rodando menos riscos o servidor terá;

b) ataques no nível da aplicação: tiram proveito dos softwares que são desenvolvidos com baixo foco em segurança, como também poucos testes na busca e avaliação de erros;

c) ataques com scripts e programas exemplos;

d) ataques a serviços mal configurados: mesmo softwares que possuem certo grau de segurança podem ser usados em ataques pela má configuração dos mesmos, por administradores mal informados ou por displicência.

5.2.3.2 Elevação de Privilégio

O último objetivo de um intruso é obter acesso de *root* ao sistema. Em alguns casos ele pode obter diretamente, mas na maioria dos casos ele possui acesso restrito e vai à busca do *root* para acesso completo.

5.2.3.3 Enviando Programas

Após ganhar acesso o intruso geralmente necessita de seus softwares particulares para executar diferentes ações, como: elevar privilégio, manter acesso, escutar a rede ou limpar seus rastros. Para tanto o mesmo faz o *upload* destes softwares para o servidor invadido para depois usá-los.

5.2.3.4 Baixando Dados

Para poder analisar com mais calma os dados do sistema invadido, como documentos e arquivos de configurações do sistema o intruso acaba fazendo o *download* destes, para poder analisá-los *off-line*.

5.2.3.5 Mantendo Acesso

Depois de ter acessado o sistema o intruso normalmente vai utilizar de meios para poder entrar quantas vezes quiser no sistema e desta vez de uma

maneira mais facilitada. Para tanto utiliza de artifícios como: portas dos fundos (*backdoors*) e cavalos de tróia (*trojans*). Tanto um quanto o outro, são facilitadores de entrada no sistema, a diferença entre eles é que, por exemplo, uma *backdoor* pode até mesmo ser uma conta adicionada na lista de usuários e *trojans* são programas legais que alterados e acionados executam uma determinada função, por exemplo: um *ls* alterado poderia disponibilizar uma *shell* em uma determinada porta. Um *trojan* não deixa de ser uma *backdoor*.

5.2.3.6 Encobrindo Rastros

Certamente um intruso após suas atividades tenta limpar seus rastros, para não deixar possíveis provas para os investigadores. A maneira mais básica é a exclusão ou a limpeza dos arquivos de *logs* do sistema e processos, como também a exclusão de arquivos utilizados para o acesso.

5.3 O ATAQUE

Após ter uma visão geral de como funciona basicamente um processo de intrusão, será demonstrado algumas técnicas públicas utilizadas por intrusos levando em consideração o enfoque da presente pesquisa. Deixando de lado técnicas utilizadas de acesso ao computador e enfocando técnicas utilizadas no computador, como serviços e o sistema operacional.

5.3.1 Softwares Vulneráveis

A maioria dos programas não são projetados, criados e implementados por profissionais que priorizem o desenvolvimento de softwares seguros (FOSTER, 2005). Por causa disso, muitos softwares vulneráveis estão presentes em servidores. Abaixo, algumas das vulnerabilidades que podem ser encontradas nos softwares:

5.3.1.1 Estouro de Buffer (Buffer overflow)

Ocorre quando um buffer que possui espaço de memória limitado recebe dados maiores que o esperado. Existem duas classes distintas: *Stack overflow* e *Heap overflow*.

5.3.1.2 Estouro de Pilha (Stack overflow)

É a técnica mais comum de *buffer overflow*. Ocorre quando o *buffer* que reside na área de pilha de um processo esta preparado para receber um determinado tamanho de dado e propositalmente recebe um tamanho maior, que obviamente não consegue tratar. Os dados extras estouram a pilha e acabam sobrescrevendo outras regiões da memória, onde podem ser introduzidos novas instruções, dando o controle do processo ao intruso. Normalmente o endereço de retorno é sobrescrito o que permite a especificação do endereço de uma *shell*, na

qual o intruso terá um *prompt* e poderá executar comandos no sistema vulnerável. Os *softwares* são vulneráveis pelo tratamento incorreto das funções de string como *strcpy*, *strcat* e assim por diante.

5.3.1.3 Estouro de Heap (Heap overflow)

A área de *heap* refere-se à memória que é dinamicamente alocada por uma aplicação para armazenagem de variáveis. No *heap overflow*, o intruso tenta sobrescrever variáveis como senhas, nome de arquivos e UIDs no *heap*. A diferença entre *stack* e *heap overflow* é que o *stack overflow* especifica um outro endereço de retorno trocando o fluxo de execução do programa, já o *heap overflow* propõem-se a aumentar o nível de privilégios do sistema sobrescrevendo as variáveis dinâmicas armazenadas pela aplicação (CHUVAKIN, PEIKARI: 2004).

Outras formas públicas de ataque a softwares ainda podem ser citadas, como: *format string*, *integer overflow*, *race conditions* e outros.

Para tirar proveito destes tipos de ataques são utilizados *softwares* chamados *exploits*, que exploram a vulnerabilidade do serviço, seja remoto ou não, normalmente permitindo o acesso a um *shell* com poderes do usuário que está rodando o serviço.

5.4 PROCESSO DE LIMPEZA DE RASTROS E PERMANÊNCIA NO SISTEMA

Após explorar e ter acesso ao sistema, o intruso acaba utilizando softwares para encobrir seus rastros e facilitar a sua permanência e futuros acessos. Nesta área o método mais conhecido e utilizado são os *softwares* chamados *rootkits*.

5.4.1 Rootkits

São ferramentas utilizadas como apoio por intrusos pós intrusão, com o intuito de facilitar o acesso futuro ao sistema, apagar possíveis rastros, esconder processos e arquivos, entre outras funcionalidades. Com o passar do tempo os *rootkits* foram evoluindo em resposta as técnicas desenvolvidas na sua detecção. De maneira geral, eles podem ser divididos em modo usuário e modo *kernel*. O primeiro manipula apenas binários e arquivos do sistema, já o segundo utiliza programação no nível do *kernel*.

5.4.1.1 Rootkits – Modo Usuário

São *rootkits* da primeira geração, que consistem na substituição de alguns programas chaves do sistema por versões maliciosas, que pode fazer sua função normalmente mas que possuem funções específicas implantadas pelo

intruso. Estas ferramentas podem conter *backdoors*, *trojans*, *sniffers*, entre outras funcionalidades dependendo do intruso. Por exemplo, alteração do comando *ls* para que não mostre arquivos específicos, ou a implantação de uma backdoor no *sshd* que libera uma *shell* ao conectar numa porta específica.

Abaixo uma lista de ações que os intrusos podem tomar e possíveis programas que possam ser substituídos:

a) esconder arquivos: du, find, sync, ls, df, lsof, netstat;

b) esconder processos: killall, pidof, os, top, lsof;

c) sniffer e aquisição de dados: ifconfig, passwd;

d) esconder conexões: netstat, tcpd, lsof, route, arp;

e) executar tarefas: crontab, reboot, halt, shutdown;

f) esconder *logs*: w, who, last;

g) backdoors: inetd, login, rlogin, rshd, telnetd, sshd, su, chfin, passwd, chsh, sudo.

Exemplo de *rootkits* conhecidos: T0rnkit, LKR (The Linux Rootkit), TrojanIT, Lrk5, e muitos outros.

5.4.1.2 Rootkits – Modo Kernel

Com o passar do tempo novas técnicas utilizadas em *rootkits* foram sendo implementadas, este tipo de *rootkits* podem tirar todas as vantagens de

estarem executando em modo *kernel*, como por exemplo: esconder arquivos e pastas sem precisar modificar e transferir vários programas, esconder processos e placa de redes em modo promíscuo, redirecionamento de execução de programas, como ainda manipular dispositivos.

A primeira técnica, considerada da segunda geração de *rootkits*, consiste na utilização de módulos do *kernel* por intrusos com a função de manipular funções do sistema. Um exemplo clássico de *rootkit* que utiliza esta técnica é o *Adore*.

A segunda técnica não utiliza módulos do *kernel* para efetuar mudanças na sua funcionalidade, desenvolvida em casos onde o administrador do sistema retira o suporte a módulos do *kernel*. Nesta técnica o intruso manipula diretamente através da memória onde esta carregada o *kernel*, através do dispositivo */dev/kmem*. Ou seja, podem implementar qualquer funcionalidade já mencionadas dinamicamente através da leitura e inserção de códigos diretamente no espaço de memória do *kernel*. Exemplo de *rootkit* que utiliza esta técnica: SuckIT.

As técnicas e métodos apresentados anteriormente, podem variar de acordo com o sistema e com os objetivos do intruso. Assim como os *rootkits*, outras ferramentas e métodos vem se adaptando a medida que novos procedimentos de segurança são aplicadas na detecção de intrusos. Por isso a necessidade dos profissionais da área estar sempre atualizados e tentar “pensar como um intruso”.

Cada ação do intruso pode gerar de alguma forma interferências em outras áreas do sistema, que podem servir como provas se as evidências forem capturadas e analisadas de forma adequada. Em um ambiente GNU/Linux, por exemplo, possibilita o perito analisar vários itens capturados do sistema na busca de provas.

6 EVIDÊNCIAS EM SERVIDORES GNU/LINUX: FONTE E ANÁLISE

Tendo uma visão mais apurada de como funciona o processo de um ataque, o perito forense deve ter em mente as conseqüências dos possíveis passos do intruso, levando em consideração que cada ação pode interferir no sistema atacado. Dessa forma, o perito pode prever ou imaginar os principais repositórios de evidências para que possa analisá-las. A seguir, um exame mais detalhado dos possíveis locais onde podem ser encontradas e analisadas as evidências nos servidores GNU/Linux.

6.1 PROCESSOS

A análise dos processos em execução na perícia forense é de extrema importância, através deles pode-se, por exemplo, localizar os serviços que estão sendo disponibilizados pelo servidor, como também localizar processos maliciosos. Dependendo da análise efetuada, é possível detectar com clareza uma intrusão, que poderia ser utilizado como prova em um processo. Por exemplo, na presença de processos em execução de *rootkits*, *backdoors* ou *trojans*, tem-se a certeza de que o sistema foi comprometido.

Uma lista completa de todos os processos rodando no sistema, como também a identificação da função de cada um é necessário para que se possam determinar quais processos são maliciosos ou não. Caso sejam detectados, pode-se

determinar o modo de agir do criminoso e para que propósito o servidor esteja sendo utilizado.

Abaixo algumas “pistas” que podem ser encontradas na lista de processos capturadas do sistema:

a) processos com nome ou comandos de execução suspeitos;

b) processos com conexão aberta com algum *host* ou esperando conexões em portas TCP/UDP suspeitas;

c) processos executados por usuários sem privilégios ou até mesmo “inexistentes”;

d) processos ausentes, mas que deveriam estar executando;

e) processos utilizando recursos incomuns.

Algumas ferramentas são disponibilizadas por padrão em diversas distribuições que podem ajudar o perito na aquisição destas informações, são elas:

6.1.1 Ferramenta PS

O *ps* pode ser utilizado para listar todos os processos do sistema, independente do seu estado. Os dados coletados por esta ferramenta são de grande importância numa investigação já que podemos coletar várias informações do processo como: nome do usuário e PID do dono, porcentagem utilizada de CPU, tamanho ocupado na memória, prioridade de processamento, hora de início, comando utilizado para execução, como também o estado do processo

(executando, espera em disco, suspenso, interrompido ou zumbi), entre outras informações.

Como a ferramenta *ps* carrega os valores vindos da interface */proc*, os dados gerados por ela podem ter sido manipulados, já que o sistema pode ter sido comprometido em nível do *kernel*. Por este motivo, a saída do *ps* deve ser utilizada como fonte de comparação entre as saídas de outras ferramentas do mesmo propósito, assim é possível detectar discrepâncias e determinar processos maliciosos executando no sistema.

6.1.2 Ferramenta Lsof

O *lsof* é uma ferramenta que possibilita o perito listar todos os arquivos ligados ao processo. Estes arquivos podem ser: arquivos regulares, diretórios, sockets, bibliotecas e outros. O comando *lsof* também é sensível a ataques em nível do *kernel*.

6.1.3 Ferramenta Strace

Strace é um utilitário que intercepta e grava todas as chamadas de sistemas invocadas e de todos os sinais recebidos por um processo. Esta ferramenta é útil para o rastreamento das chamadas de sistemas, verificando se tiveram alguma alteração.

6.1.4 Sistema de Arquivos */proc*

O sistema de arquivos */proc* é uma interface de acesso aos parâmetros do sistema, disponibilizado pelo *kernel* em um sistema hierárquico, dividido em subdiretórios correspondente aos números PID dos processos, no qual pode-se navegar normalmente como se fosse um sistema de arquivos normal. Mas que não possui acesso a disco, os dados ficam na memória e são gerados dinamicamente pelo *kernel* (MANDIA, PROSISE:2002), (WELSH: 2002), (FLICKENGER, 2003). Inúmeras informações podem ser adquiridas no */proc*, entre elas: informações de processos rodando, configuração de CPU, hardware, arquivos abertos, mapeamento de memória, entre outras.

Cada processo possui seus subdiretórios com informações do processo:

a) *cmdline*: mostra a linha de comando e seus argumentos para execução do processo. Caso o processo esteja todo na área de *swap* (processo zumbi), a *cmdline* não retorna a linha de comando.

b) *cwd*: *link* para o diretório atual de trabalho do processo.

c) *environ*: contém as variáveis de ambiente no qual o processo esta sendo executado.

d) *exe*: *link* ao binário do processo. Mesmo que o arquivo tenha sido excluído é possível recuperá-lo através deste *link*.

e) *fd*: este subdiretório contém um *link* simbólico para todos os arquivos abertos pelo processo. Também pode ser recuperado assim como o binário.

f) *maps*: contém o mapa atual de regiões da memória e suas permissões de acesso.

g) *mem*: este não é igual ao dispositivo *mem*, apesar de ter o mesmo número de dispositivos. O dispositivo */dev/mem* é a memória física antes da conversão de endereços, mas o arquivo *mem* aqui descrito é a memória acessada pelo processo.

h) *mounts*: pontos de montagem do sistema.

i) *root*: o Linux suporta a idéia de uma raiz de sistema de arquivos por processo, definidos pela chamada de sistema *chroot*.

j) *stat*: informações sobre o *status* do processo. Isso é fornecido por *ps*.

Além das informações dos processos podem-se listar ainda outras informações úteis que o */proc* pode gerar:

a) *kcore*: arquivo do tipo *core* que representa a memória física do sistema, o tamanho do arquivo é do tamanho da memória RAM do sistema.

b) *ksyms*: Contém as definições dos símbolos exportados pelo *kernel* usados pelas ferramentas de módulos para dinamicamente ligar e vincular módulos carregáveis.

c) *meminfo*: informa a quantidade de memória livre e utilizada (tanto a memória física como a de troca) assim como a memória compartilhada e os *buffers* usados pelo *kernel*.

d) *modules*: lista os módulos carregados pelo sistema.

e) *version*: identifica a versão do *kernel* que está sendo executada.

Muitas ferramentas forenses utilizam a interface */proc* para obter informações do sistema, como já mencionado, desta forma quaisquer mudanças nas chamadas de sistemas estas informações poderão conter alterações de acordo com o objetivo do intruso.

6.1.5 Diagnóstico de um Processo

Para uma análise mais detalhada do processo, além dos dados mencionados anteriormente, outros mais devem ser coletados e analisados para se ter um diagnóstico mais apurado. Estes novos dados e a ligação entre eles é que possibilita traçar o perfil e a função do processo em um formato mais fiel à realidade.

Das ferramentas mencionadas anteriormente podem ser extraídas de algumas informações temporais importantes, como tempo de execução e o início da execução, como também saber o usuário que o executou e listar todos os arquivos ligados ao processo. Estas informações somados a uma análise da linha do tempo do arquivo binário e os arquivos ligados a ele através da *time line* do sistema de arquivos, que será apresentada adiante, permitem traçar o perfil temporal do processo.

Sua funcionalidade pode ser ainda avaliada além dos arquivos abertos, através das conexões em andamento e portas na escuta do processo. Além disso, o estudo das chamadas de sistemas solicitadas através do *strace* também pode

extrair informações importantes do seu funcionamento. Descobrir os parâmetros usados para a execução do processo e as variáveis de ambiente no qual ele está rodando devem ser levadas em consideração na análise.

Dependendo do cenário encontrado, a análise mais confiável é o uso de técnicas de depuração ou engenharia reversa no binário do processo através de ferramentas específicas como *debuggers* e *disassemblers*, respectivamente. Um exemplo disso é o *gdb* que pode ser usado como depurador no caso de ter o *core file* do processo e também avaliar as linhas de código em *assembly* do binário usando o comando *disass main*, por exemplo. Outros dados sobre o binário ainda podem ser avaliados através de ferramentas, como: *strings*, *elfdump*, *objdump*, entre outros. Outra forma de verificação da funcionalidade do processo é a comparação do valor *hash* do binário com *trojans* e *backdoors* conhecidos.

A verificação da igualdade do binário sendo executado (residente na memória) com o arquivo armazenado no *hard disk* deve ser conferida antes do exame. Algumas técnicas utilizadas por intrusos consistem na infecção de binários ELF dinamicamente através da interface de memória */dev/mem*.

6.2 MÓDULOS DO KERNEL (LKM)

Aproveitando-se da capacidade do *kernel* do Linux de carregar e descarregar módulos dinamicamente. Foram desenvolvidos módulos maliciosos com propósitos diversos. Por estarem executando no modo kernel, estes módulos possuem poucas restrições e são capazes, por exemplo, de interceptar e manipular as chamadas de sistemas para outros propósitos (MOHAY, 2003). As utilizações

mais comuns de ataques a LKM são *rootkits* que são capazes de esconder processos, arquivos e serviços, cujo objetivo principal é esconder rastros e garantir a permanência do intruso (MANDIA, PROSISE:2002). E como *keyloggers*, que gravam tudo o que é digitado e podem ser acessado mais tarde.

Estas ferramentas não são muito fáceis de serem detectadas já que manipulam as chamadas de sistemas, fornecem informações falsas e são especificamente projetadas para prevenir sua detecção (MANDIA, PROSISE: 2002). Em alguns casos, a utilização em conjunto de comandos ou ferramentas coletando as mesmas informações e a comparação entre as saídas pode-se detectar anomalias ou discrepâncias.

Algumas ferramentas e procedimentos que podem ser utilizadas na detecção:

a) lsmod: é um utilitário que carrega e formata de maneira amigável a saída da interface */proc/modules*.

b) interface */proc/modules*: visualizar a saída vinda da interface disponibilizada pelo kernel. Lembrando que o */proc* pode ter sua saída alterada caso o intruso tenha algum modulo malicioso instalado que altere ou faça um redirecionamento (*hook*) nas chamadas de sistemas.

6.3 MEMÓRIA PRINCIPAL

A memória principal, mais conhecida como memória RAM é onde ficam armazenados os principais dados voláteis do sistema, como: informações

dos processos, conteúdo de arquivos abertos, dados que ainda não foram gravados em disco (*cache*), como também informações do sistema operacional.

O conteúdo da memória RAM é um arranjo não contíguo no qual a reconstrução completa requer um conhecimento aprofundado. Este conteúdo é utilizado normalmente através de buscas por *strings*, que podem conter senhas e conteúdo de um arquivo aberto recentemente (MANDIA, PROSISE: 2002).

A análise da memória pode ser feita através do *dump* da mesma gerada através da interface */proc/kcore*. O resultado do *dump* é um arquivo do mesmo tamanho da memória RAM no formato ELF, que pode ser manipulado pelo *gdb* ou qualquer outra ferramenta específica. Outra forma de geração de *dump* da memória acontece quando ocorre uma falha no sistema, neste caso um arquivo denominado *crash dump* é gerado. Este arquivo também contém a imagem da memória do momento do *crash* do sistema.

Da mesma forma acontece com os processos que se deseja examinar isoladamente. Um processo ao receber determinados sinais, por exemplo, acesso a área de memória não autorizada ou ainda um sinal definido pelo usuário dono do processo, pode gerar uma imagem com informações a respeito do aplicativo e seu estado, para que depois possa ser depurado (MAXWELL, 2000). Estas imagens são chamadas de *core files* ou *core dump*.

A busca e a análise de *core files* é importante na detecção de possíveis falhas e ataques na busca de pistas da intrusão, como em um ataque de *buffer overflow*. O estudo do *core file* pode determinar de que forma o ataque ocorreu.

Como já mencionado, a memória contém muitas informações importantes sobre o sistema. Uma análise detalhada é necessária, já que nela podem residir informações sensíveis, não disponíveis na memória secundária. Um exemplo disso é a técnica anti forense que consiste na execução do binário em um sistema remoto sem a criação de arquivos no *hard disk*. Neste tipo de ataque, por exemplo, a análise da memória do sistema é imprescindível, já que neste tipo de ataque não é gerado nenhum tipo de evidência no sistema de arquivos.

A análise da memória entre outras informações pode determinar a lista de processos sendo executados, valores de registrados, tabelas, entre outros. Além de poder detectar *rootkits* que utilizem módulos do *kernel*.

6.4 SISTEMA DE ARQUIVOS

O projeto dos sistemas operacionais da família Unix, é centralizado no sistema de arquivos (BOVET, CESATI: 2002). É a parte mais visível do usuário e pode conter os mais diversos tipos de dados, como: músicas, documentos, e-mails, fotos que podem estar relacionados com a investigação. Além disso, é no sistema de arquivos que o Linux armazena seus arquivos de configurações, repositório de arquivos temporários, memória virtual e informações mais sensíveis como representações de processos e memória.

A busca e análise de evidências no sistema de arquivos são muito valiosas, já que qualquer atividade do usuário no sistema pode produzir uma pista sobre suas atividades (CASEY, 2004). Por exemplo, aplicações podem gerar arquivos temporários ou utilizar memória virtual (*swap*) que acaba gerando dados

úteis para análise no *hard disk*. Como também qualquer acesso ou manipulação de arquivo ou diretório, acaba alterando seus respectivos *time stamps*, o que possibilita a verificação da data e hora da criação, do último acesso, alteração, até mesmo exclusão dos arquivos.

Em muitos casos, dados escondidos no *hard disk* podem conter evidências valiosas. Estes dados escondidos podem ser fragmentos de arquivos excluídos, sobras de dados de um *hard disk* particionado, ou arquivos propositalmente ocultados pelo intruso. Para ocultar dados no disco, os intrusos utilizam técnicas que se aproveitam da manipulação do próprio sistema de arquivos.

A seguir serão demonstradas algumas das técnicas públicas utilizadas por intrusos na ocultação de dados em um *hard disk*.

6.4.1 Host Protected Area (HPA)

É uma área especial do disco que pode ser usado para salvarem dados. Ele foi adicionado no ATA-4 por fabricantes para armazenar seus dados e que ficassem de forma protegida, sem que o usuário apagasse quando formatasse o *hard disk*. Por padrão, muitos discos vem com o tamanho HPA como 0 bytes (CARRIER, 2005). Sabendo disso, intrusos podem esconder dados nesta área quando ela existir. Para verificar se existe HPA no disco basta apenas comparar o número total de setores no disco com o número total de setores que o usuário pode acessar. O kit TSK possui a ferramenta *diskstat*, que enumera estes valores para o perito.

6.4.2 Sistema de Arquivos

A estrutura de dados que descreve o super bloco possui tamanho de 1,024 bytes e fica localizada a um *off-set* de 1,024 do início do sistema de arquivos. Este espaço entre o super bloco e início do sistema de arquivos é reservado para códigos de boot e podem conter dados escondidos. Da mesma forma acontecem com o super bloco e a tabela descritora de grupos que podem conter dados escondidos ao final da estrutura de dados, já que as mesmas nem sempre utilizam todo o espaço que possuem.

6.4.2.1 Conteúdo

O setor é a menor unidade de leitura e gravação do disco e pode ter de 512 a 4096 bytes (OLIVEIRA, CARISSIMI, TOSCANI: 2004). Ao ser gravado no disco um arquivo de tamanho 612 bytes, será utilizado dois setores, sendo que o segundo ficará uma lacuna de 412 bytes. Este espaço não utilizado chama-se *slack space* e podem ser manipulados por ferramentas publicas como *bmap*. Algumas versões do kernel não deixam criar *slack space* já que preenchem os dados com zero.

6.4.2.2 Meta Dados

Alguns *inodes* são tipicamente reservados, entre os endereços de 1 a 10. O *inode* 2 por exemplo é usado pelo diretório *root*, já o *inode* 1 gerencia *bad blocks*. O *inode* 11 é o primeiro que não possui significado especial e pode ser usado para arquivos de usuário. Sabendo disso, intrusos podem utilizar alguns destes *inodes* que não estejam sendo utilizados pelo SO, assumindo que algumas ferramentas não averiguam *inodes* reservados. O kit TSK possui a ferramenta *ils*, que possibilita esta análise acessando diretamente o *inode* especificado.

6.4.2.3 Nome de Arquivos

Ao ser excluído, um arquivo no qual um ou mais processos estejam sendo utilizados, o *link count* do *inode* vai para 0, o *inode* não é liberado e é adicionado a uma lista dentro do super bloco. Mesmo não liberado, o arquivo não pode ser visto com um *ls* por exemplo. Dessa forma, alguns intrusos tentam enganar o perito. A ferramenta *fsstat* do TSK pode ser utilizado para investigar estes *inodes*.

6.4.3 Análise Propriamente Dita

6.4.3.1 Sistema de Arquivos

A análise desta categoria é essencial para as análises posteriores ao sistema de arquivos. É nesta categoria onde se pode determinar o *layout* do sistema de arquivos, já que ela armazena o tamanho e localizações para outras estruturas de dados chaves. Outro item que podemos achar são as tabelas descritores de grupos, que contém o apontamento para os grupos e pode localizar as estruturas de dados que determina a situação dos blocos e *inodes*. Usando estes dois itens, podem-se localizar todas as estruturas usadas pelos arquivos e diretórios do sistema. O super bloco fica localizado a 1,024 bytes do inicio do sistema de arquivos e seu tamanho é de 1,024 bytes.

6.4.3.2 Conteúdo

Arquivos e diretórios são armazenados em blocos de dados e localizá-los é fundamental na investigação já que é nos blocos que residem o conteúdo do arquivo. Além de localizar e ler o conteúdo, também se pode determinar o estado dos blocos.

Sabendo o tamanho dos blocos que fica no super bloco, é possível encontrar qualquer bloco de dados já que o bloco reside no primeiro setor do sistema de arquivos.

O estado de alocação de um bloco é dado pelo mapa de bits do bloco, que possui um apontador no descritor de grupos, que por sua vez pode ser encontrado no bloco 1 do grupo. Cada bit do mapa de bits do bloco corresponde a um bloco dentro do grupo, o que significa que num sistema de arquivos onde o tamanho do bloco é de 4,096 bytes, o total de blocos por grupo que um mapa de bits pode endereçar é de 32,768 (4,096 * 8) blocos.

6.4.3.3 Meta Dado

Para examinar com detalhes um arquivo ou diretório é necessário processar os dados de um *inode*. Localizar um *inode* é preciso saber em qual grupo de blocos ele pertence. Logo após, processar o descritor do grupo para encontrar o mapa de bits e por último identificar que entrada na tabela o *inode* em questão esta e analisá-lo.

O estado de alocação de um *inode* pode ser identificado pela análise do mapa de bits do *inode*, no qual é determinado pelo descritor de grupo. Caso o bit seja 1, então o *inode* esta alocado.

Esta análise apurada dos *inodes* não deve ser restrita apenas aos alocados, os excluídos podem conter dados temporais importantes na investigação e podem identificar, por exemplo, quando um arquivo foi excluído.

6.4.3.4 Nome de Arquivos

Através da estrutura de dados das entradas de diretório, podem-se listar todos os diretórios e seus subdiretórios e arquivos. Dando a possibilidade de navegação entre a árvore de diretórios, assim como procurar por um arquivo em específico.

A análise consiste em achar uma entrada de diretório válida, localizar as próximas entradas e acessá-las sequencialmente. A primeira entrada é o *root* que possui o *inode* fixo igual a 2. Caso o investigador queira encontrar somente arquivos e diretórios alocados, o processo consiste em somar sua posição atual com o campo *rec_len*, que é uma espécie de apontador para a próxima entrada válida. Ao procurar também por *inodes* "desalocados" é preciso desprezar o campo *rec_len* e calcular o tamanho que esta entrada teria realmente, já que o campo *rec_len* pode ter sido alterado para apagar outras entradas de diretório, conforme mencionado sobre o processo de exclusão em capítulos anteriores.

O sistema operacional possui várias funcionalidades e componentes nos quais estão interligados. Por isso a análise das evidências extraídas dos pontos chave, citados no decorrer do capítulo, além de serem examinados individualmente também devem ser estudados como membros participantes de um contexto, onde seus componentes interagem entre si. Neste formato, o investigador pode ter uma visão geral dos acontecimentos.

Na prática, a complexidade da análise das evidências depende diretamente da quantidade e qualidade dos dados coletados pelas primeiras respostas, no qual esta sujeito ao ambiente encontrado pelo perito e também do nível do intruso que pode utilizar técnicas denominadas anti-forense. Se o computador estiver desligado, por exemplo, a investigação fica praticamente restrita a análise do sistema de arquivos.

7 TRABALHOS CORRELATOS

Durante os estudos objetivando esta pesquisa, seja na proposta ou no seu desenvolvimento, foram analisados alguns trabalhos com propósitos semelhantes, mas cujo foco era outro. Abaixo a descrição de alguns trabalhos escolhidos, envolvendo a perícia forense computacional.

O primeiro trabalho intitulado: “Aplicação de Técnicas de Forense Computacional e Respostas a Incidentes na Internet”, pretendente ao título de especialização *Lato Sensu*, tem como objetivo estudar metodologias e técnicas forenses aplicadas à segurança em redes. No qual se conclui que com o número elevado de áreas torna a ciência forense computacional complexa, e para uma boa perícia, o investigador deve estar focado apenas nos seus objetivos e que sua experiência e qualidade são importantes, mas que devem ser complementadas com boas ferramentas e uma excelente metodologia. O trabalho em questão pode ser encontrado, no endereço: http://www.modulo.com.br/pt/page_i.jsp?page=3&catid=17&objid=69&pagenu mber=0&idiom=0.

O segundo trabalho escolhido, pretendente ao título de *Lato Sensu*, intitulado “Perícia Forense Aplicada à Informática”. Tem como meta abordar e explicar o tema perícia forense, com um trabalho prático específico voltado à análise de *logs*, do servidor de páginas de internet Microsoft IIS (*Internet Information Server*). Os resultados obtidos ficaram de acordo com a proposta do trabalho. O trabalho em questão pode ser encontrado, no endereço: http://www.secforum.com.br/textos/andrey-freitas.pdf.

8 PERÍCIA FORENSE EM SERVIDORES GNU/LINUX

O presente capítulo apresenta o trabalho de pesquisa proposto propriamente dito, onde será demonstrada na sua grande maioria de forma prática a teoria abordada até então. O processo prático possui o intuito de demonstrar da maneira mais real possível, de acordo com as possibilidades e limitações existentes, a coleta e a análise de evidências em um ambiente GNU/Linux, sendo o segundo item o foco do trabalho. A escolha do trabalho prático é justificada primeiramente por complementar toda a base teórica apresentada e principalmente para expor as reais dificuldades encontradas nas tarefas executadas, nos quais as literaturas estudadas se eximiram.

A parte prática foi divida basicamente em três etapas: processo de intrusão, coleta e análise das evidências. A finalidade da intrusão é meramente ilustrativa, uma simulação básica que serve como um parâmetro de comparação do que pode ser resgatado ou não durante a investigação. O objetivo não é responder as perguntas: quem, como, onde e por quê?, mas sim analisar as limitações e possibilidades numa investigação.

Para a parte prática foi utilizado um computador doméstico, no qual possui 1 (um) hard disk com duas partições Ext2, o primeiro com 878,5 MBytes utilizado como servidor e o segundo com aproximadamente 2 GBytes utilizado para a análise forense. Foram instaladas em ambas as partições a distribuição Debian GNU/Linux 3.1 r2 com o Kernel 2.4.27. Para a primeira partição foram instalados os pacotes básicos para um servidor GNU/Linux, rodando Apache2 e Php4. A segunda partição foi apenas instalada os pacotes básicos.

Neste formato de configuração, o computador pode ser utilizado tanto como servidor quanto como ferramenta para a perícia forense, bastando apenas selecionar a partição correta na inicialização do sistema.

Antes de prosseguir com o conteúdo, mais alguns esclarecimentos são necessários para o bom entendimento do trabalho quanto a apresentação de comandos e saídas:

a) os comandos apresentados a seguir na sua maioria, utilizam o redirecionamento de saída através do comando “>”. Significando que a saída que deveria ir para o vídeo, é gravada em um arquivo de texto. Para uma melhor apresentação da pesquisa, o comando “redirecionador” será muitas vezes ocultado e mostrado no texto sua saída;

b) algumas saídas estão resumidas e a mesma completa deve ser verificada no CD-ROM em anexo;

c) O caminho no qual o comando esta sendo executado será ocultado, mostrando somente o destino. Partindo a premissa que o comando esta sendo executado do CD-ROM: #/mnt/cdrom/.

8.1 PROCESSO DE INTRUSÃO

A simulação de intrusão começa quando o “suposto” intruso encontra uma possibilidade real de invasão ao navegar pelo site, no qual fica hospedado no

servidor a ser analisado. A falha encontrada consiste numa página PHP no qual utiliza inapropriadamente "*include*" de outras páginas, sem nenhuma verificação da origem das mesmas. Neste cenário o invasor pode chamar sua própria página PHP hospedada em outro servidor, e seu script (linhas de código) será executado como se fizesse parte da página no qual esta chamando. Esta técnica é denominada de XSS (*Cross Site Scripting*).

O script utilizado possui a função *system* presente no PHP, que executa qualquer comando no servidor como se estivesse num *shell*. Obviamente, os comandos executados pelo site possuem as restrições atreladas ao usuário no qual está rodando o serviço, no caso o Apache. O formato do ataque fica assim:

http://localhost/index.php?pagina=http://www.xxx.com/meuscript.php

Após conseguir executar algum comando qualquer no servidor, o intruso abre uma conexão direta através da ferramenta *netcat* presente no servidor. O *netcat* pode abrir portas nos qual o servidor ficará "escutando" a espera de conexão. Logo após o intruso pode acessar o servidor com um *telnet*, por exemplo. Caso o *netcat* não estivesse instalado no servidor o invasor poderia facilmente fazer o *download* do mesmo, através do comando "*wget* http://www.yyy.com/nc".

Agora conectado ao servidor e com um *shell* válido o intruso começa o processo de elevação de privilégios. No caso, começou pelo *download* de um *exploit* chamado "*exp*". Executando o *exploit*, o intruso conseguiu acesso *root* e baixou um *rootkit* chamado *Adore*. No qual descompactou e renomeou a pasta para *pages*, numa tentativa de esconder os arquivos no meio de outras do site.

8.2 COLETA DE EVIDÊNCIAS

Como já mencionado em capítulos anterior, a coleta das evidências devem ser exata, ou seja, não devem sofrer qualquer tipo de alteração durante o processo de aquisição e armazenagem das mesmas.

Em um ambiente de intrusão real, os dados são enviados da máquina analisada para um outro computador remotamente através da rede. Desta forma, garante um impacto mínimo no sistema a ser examinado. No formato do trabalho proposto, a coleta das evidências será feita diretamente entre as partições, como os sistemas são executados de forma independente, os mesmos também não sofrem quaisquer alterações.

Ao se coletar evidências em um sistema, a aquisição sempre que possível deve ser feita do mais volátil ao menos volátil (VACCA, 2002). De acordo com a RFC 3227 e o propósito do trabalho, a lista fica assim:

a) memória principal do sistema;

b) estado do sistema operacional (processos em execução, módulos do sistema, usuários “logados”, entre outros);

c) dispositivos de armazenagem secundária.

8.2.1 Preparação

Primeiramente deve ser criado um CD-ROM com todas as ferramentas que serão utilizadas na aquisição dos dados. As ferramentas devem ser compiladas estaticamente, preferencialmente na mesma arquitetura do sistema a ser analisado,

para que as mesmas rodem perfeitamente e sem a necessidade da utilização de bibliotecas do sistema.

Com o CD-ROM pronto, o mesmo deve ser “montado” no computador vítima e ser executado um *shell* confiável para poder capturar os dados do sistema analisado.

```
#mount /dev/hdd /mnt/cdrom
#/mnt/cdrom/bash
```

8.2.2 Coleta de Dados Voláteis

a) Gerando a imagem da memória do sistema e do kernel, respectivamente:

```
# dd if=/dev/mem of=/mnt/forensic/tcc/evidencias/memoria
# dd if=/proc/kcore of=/mnt/forensic/tcc/evidencias/kcore.dd
```

b) Obtendo informações gerais referentes ao armazenamento secundário do sistema:

```
#grep hd /var/log/dmesg
```

c) Sabendo que o computador possui apenas um hard disk, é possível capturar os particionamentos do mesmo:

```
#fdisk –l -uS /dev/hda
Disco /dev/hda: 9733 cilindros, 255 cabeças, 63 setores/trilha
Unidades = setores de 512 bytes, contando a partir de 0

   Device Boot    Start       End   #sectors  Id  System
/dev/hda1            63  62139419   62139357  2d  Desconhecido
/dev/hda2      62139420 156360644   94221225   f  W95 Ext'd (LBA)
/dev/hda3             0         -          0   0  Vazia
/dev/hda4             0         -          0   0  Vazia
/dev/hda5      62139483  92871764   30732282  2d  Desconhecido
/dev/hda6      92871828 150223814   57351987  2d  Desconhecido
/dev/hda7     150223878 154320389    4096512  2d  Linux
/dev/hda8     154320453 156119669    1799217  83  Linux
/dev/hda9     156119733 156360644     240912  2d  Desconhecido
```

d) Capturando dados gerais do sistema:

```
#uname –a
Linux server 2.4.27 #1 SMP Seg Jun 12 14:54:25 BRT 2006 i686 GNU/Linux

#uptime
19:29:29 up  3:08,  4 users,  load average: 0.00, 0.01, 0.00

# date
Seg Jun 12 19:29:40 BRT 2006
```

e) Determinando os usuários logados ao sistema:

```
#w
23:57:58 up 8 min,  1 user,  load average: 0,00, 0,03, 0,02
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1     -                23:50    0.00s  0.22s  0.00s w
```

f) Listando os processos em execução:

```
#ps aux
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2   1492  500 ?        S    Jun12   0:00 init [2]
root         2  0.0  0.0      0    0 ?        S    Jun12   0:00 [keventd]
root         3  0.0  0.0      0    0 ?        SN   Jun12   0:00 [ksoftirqd_CPU0]
root         4  0.0  0.0      0    0 ?        S    Jun12   0:00 [kswapd]
lp         955  0.0  0.4   2452  864 ?        Ss   Jun12   0:00 /usr/sbin/lpd -s
root       962  0.0  0.8   3720 1544 ?        Ss   Jun12   0:00 /usr/sbin/sshd
root       967  0.0  0.4   2368  932 ?        Ss   Jun12   0:00 /sbin/rpc.statd
```

```
daemon     970  0.0  0.3  1672  636 ?        Ss   Jun12   0:00 /usr/sbin/atd
root       973  0.0  0.4  1756  808 ?        Ss   Jun12   0:00 /usr/sbin/cron
root       979  0.0  4.1 16424 7924 ?        Ss   Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
root       990  0.0  0.2  1484  472 tty6     Ss+  Jun12   0:00 /sbin/getty 38400 tty6
www-data   991  0.0  4.1 16424 8016 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
www-data   992  0.0  4.2 16428 8224 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
root      1018  0.5  1.5  5916 3056 tty2     S+   Jun12   0:00 lynx localhost
www-data  1019  0.0  0.3  2224  740 ?        S    Jun12   0:00 nc -l -p 5600 -e /bin/bash
www-data  1020  0.0  4.1 16424 7944 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
```

g) Listando arquivos abertos pelos processos:

```
# lsof
COMMAND    PID       USER   FD      TYPE     DEVICE    SIZE     NODE NAME
init         1       root  cwd       DIR        3,8    1024        2 /
init         1       root  rtd       DIR        3,8    1024        2 /
init         1       root  txt       REG        3,8   31432   221212 /sbin/init
init         1       root  mem       REG        3,8   90248   204819 /lib/ld-2.3.2.so
init         1       root  mem       REG        3,8 1244752   204825 /lib/libc-2.3.2.so
init         1       root   10u     FIFO        3,8          184442 /dev/initctl
keventd      2       root  cwd       DIR        3,8    1024        2 /
keventd      2       root  rtd       DIR        3,8    1024        2 /
keventd      2       root  txt   unknown                          /proc/2/exe
keventd      2       root   10u     FIFO        3,8          184442 /dev/initctl
khubd      461       root  rtd       DIR        3,8    1024        2 /
khubd      461       root  txt   unknown                          /proc/461/exe
khubd      461       root   10u     FIFO        3,8          184442 /dev/initctl
```

h) Determinando quais portas estão abertas do sistema:

```
#netstat –na
Conexões Internet Ativas (servidores e estabelecidas)
Proto Recv-Q Send-Q Endereço Local          Endereço Remoto        Estado
tcp        0      0 0.0.0.0:5600            0.0.0.0:*              OUÇA
tcp        0      0 0.0.0.0:111             0.0.0.0:*              OUÇA
tcp        0      0 0.0.0.0:80              0.0.0.0:*              OUÇA
tcp        0      0 0.0.0.0:113             0.0.0.0:*              OUÇA
tcp        0      0 0.0.0.0:725             0.0.0.0:*              OUÇA
tcp        0      0 0.0.0.0:22              0.0.0.0:*              OUÇA
tcp        0      0 0.0.0.0:25              0.0.0.0:*              OUÇA
tcp        0      0 127.0.0.1:80            127.0.0.1:32769        ESTABELECIDA
tcp        0      0 127.0.0.1:32769         127.0.0.1:80           ESTABELECIDA
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 0.0.0.0:111             0.0.0.0:*
```

i) Determinando quais serviços estão com portas abertas:

```
#netstat –anpe
Conexões Internet Ativas (servidores e estabelecidas)
Proto Recv-Q Send-Q Endereço Local          Endereço Remoto        Estado      Usuário
PID/Program name
tcp        0      0 0.0.0.0:5600            0.0.0.0:*              OUÇA       33         2019       1019/nc
tcp        0      0 0.0.0.0:111             0.0.0.0:*              OUÇA       0          716        773/portmap
tcp        0      0 0.0.0.0:80              0.0.0.0:*              OUÇA       0          1132       979/apache2
tcp        0      0 0.0.0.0:113             0.0.0.0:*              OUÇA       0          950        950/inetd
tcp        0      0 0.0.0.0:725             0.0.0.0:*              OUÇA       0          1105       967/rpc.statd
tcp        0      0 0.0.0.0:22              0.0.0.0:*              OUÇA       0          1085       962/sshd
tcp        0      0 0.0.0.0:25              0.0.0.0:*              OUÇA       0          987        944/exim4
tcp        0      0 127.0.0.1:80            127.0.0.1:32769        ESTABELECIDA33        1153
992/apache2
tcp        0      0 127.0.0.1:32769         127.0.0.1:80           ESTABELECIDA0         2015
1018/lynx
udp        0      0 0.0.0.0:68              0.0.0.0:*                  0          657        768/dhclient
udp        0      0 0.0.0.0:719             0.0.0.0:*                  0          1090       967/rpc.statd
udp        0      0 0.0.0.0:722             0.0.0.0:*                  0          1098       967/rpc.statd
udp        0      0 0.0.0.0:111             0.0.0.0:*                  0          715        773/portmap
```

j) Listando módulos do kernel:

```
#lsmod
Module                  Size  Used by    Not tainted
af_packet              11048   1  (autoclean)
usb-uhci               19504   0  (unused)
usbcore                52268   1  [usb-uhci]
es1371                 23820   0  (unused)
ac97_codec             11252   0  [es1371]
soundcore               3268   4  [es1371]
gameport                1388   0  [es1371]
cdrom                  26212   0  [ide-cd]
```

```
#cat /proc/modules
Module                  Size  Used by    Not tainted
af_packet              11048   1  (autoclean)
usb-uhci               19504   0  (unused)
usbcore                52268   1  [usb-uhci]
es1371                 23820   0  (unused)
ac97_codec             11252   0  [es1371]
soundcore               3268   4  [es1371]
gameport                1388   0  [es1371]
cdrom                  26212   0  [ide-cd]
```

8.2.3 Coleta de Dados não Voláteis

Após a captura dos dados mais sensíveis, a próxima etapa é da obtenção dos dados não voláteis, ou seja, não são perdidas quando o sistema é desligado. Esta etapa consiste basicamente na duplicação do *hard disk*, no qual nesta pesquisa se resumirá apenas na partição do servidor.

A imagem gerada é uma cópia exata de todos os clusters, o que garante a cópia tanto do sistema de arquivos quanto áreas escondidas não acessíveis por ela. Esta cópia é chamada de *bitstream*. Atualmente a maneira mais comum de geração de imagem é salvar em um arquivo, que possui apenas os dados duplicados. Estas imagens são chamadas de imagem *raw* (CARRIER, 2005).

a) Copiando os dados para o arquivo:

```
# dd if=/dev/hda7 of=/tcc/imagem.dd
```

b) Fazendo um *hash* para verificação da integridade da cópia:

```
#md5sum /dev/hda7 > /tcc/hash_hd.txt
```

Agora com os dados gerados, pode-se fazer uma análise sobre as mesmas na busca de evidências ou informações que possam servir como prova.

8.3 ANÁLISE DE EVIDÊNCIAS

Agora com todos os dados gerados da etapa anterior, pode-se analisá-los. Buscando provas, ligações entre arquivos e processos, detecção de *trojans* e *backdoors*, entre outros que podem ser descobertos.

8.3.1 Sistema de Arquivos

Antes de qualquer análise a ser feita na imagem gerada no processo anterior, deve-se fazer verificação da integridade da mesma. Para tanto, pode-se comparar o valor *hash* gerado do *hard disk* na etapa de aquisição de evidências com o da imagem gerada. Dessa forma, podemos ter certeza que a imagem foi uma cópia exata.

a) Gerando o valor hash da imagem:

```
#md5sum /tcc/imagem.dd > /tcc/hash_imagem.txt
```

b) Comparando os valores abaixo, é comprovado que a cópia não foi alterada durante o processo de aquisição:

```
#cat /tcc/hash_hd.txt
a4d711f322d246969fac7f1630c3c1fb  /dev/hda7

#cat /tcc/hash_imagem.txt
a4d711f322d246969fac7f1630c3c1fb  /tcc/imagem.dd
```

Para a análise da imagem do sistema de arquivos gerada, a mesma pode ser “montada” para que o perito possa navegar pela árvore de diretórios do sistema de arquivos ou apenas pode ser copiada para uma área segura para ser feito um exame por ferramentas específicas. No primeiro caso, a imagem deve estar como somente leitura para que não seja alterado nenhum arquivo do sistema de arquivos analisado.

c) Montando a imagem:

```
# mount -t ext2 -o ro,noexec,loop /tcc/imagem.dd /mnt/server/
```

Após a montagem da imagem, o perito agora esta apto a navegar pela sua estrutura normalmente, como também utilizar ferramentas que facilitem suas buscas, como por exemplo, o comando *find*.

d) Um passo importante é gerar uma lista de todos os arquivos do sistema com seus respectivos valores *hash*:

```
#find /mnt/server/ -type f -exec md5sum {} \; > /tcc/arquivos_hash.txt
```

e) Outra lista que será útil em futuras análises são os tipos dos arquivos:

```
#find /mnt/server/ -type f -exec file {} \; > /tcc/arquivos_tipos.txt
```

Com a listagem completa dos arquivos do sistema, o perito pode usar o comando *cat* e *grep* para localizar arquivos específicos, através do tipo ou por palavras chaves, caso o mesmo já esteja procurando por algo em específico.

f) Buscando nomes de arquivos específicos, como vestígios de programas maliciosos conhecidos ou apenas palavras chaves:

```
#grep –if palavras_chave.txt /tcc/arquivos.txt > /tcc/busca_palavras.txt
```

g) Listando somente arquivos no formato JPG:

```
#cat /tcc/arquivos_tipos.txt | grep JPEG > /tcc/busca_jpg.txt
```

h) Com o comando find pode-se obter ainda a lista dos arquivos alterados e modificados nas últimas 24hs, respectivamente:

```
# find /mnt/server -atime 1 -ls > /tcc/arquivos_alterados24hs.txt
# find /mnt/server -mtime 1 -ls > /tcc/arquivos_modificados24hs.txt
```

Outra forma de descobrir arquivos suspeitos é verificar se o sistema esta de acordo com a *Filesystem Hierarchy Stardand* (FHS), especificada pela *Linux Standard Base* (LSB), que tenta padronizar a localização dos diversos tipos de arquivos e diretórios das várias distribuições existentes (ZILLI, 2003). Por exemplo, no */bin* não deve conter subdiretórios ou no */dev* que possui somente arquivos especiais e alguns arquivo MakeFile, além disso */etc* não deve conter binários já que é um diretórios de arquivos de configurações, entre outros.

i) Procurando arquivos regulares no */dev*:

```
# find /mnt/server/dev/ -type f –ls > /tcc/saida_dev.txt
```

j) Procurando por subdiretórios no */bin:*

```
# find /mnt/server/bin/ -type d –ls > /tcc/saida_bin.txt
```

k) Procurando por binários no */etc*:

```
#cat /tcc/arquivos_tipos.txt | grep ELF | grep /etc > /tcc/busca_binarios_etc.txt
```

Programas root com SUID e SGID “setados” são uma fonte valiosa para escalação de privilégio por um intruso (PROSISE, MANDIA: 2003). Quando um processo é executado o privilégio fica atrelado ao usuário que o executou, caso o binário esteja “setado” com SUID ou SGID, o processo privilegia o usuário ou o grupo (respectivamente) dono do executável.

l) Procurar binários com estas características é importante, já que pode ser uma pista de como o intruso conseguiu acesso *root* ao sistema:

```
# find / -perm -004000 -type f –print > /tcc/arquivos_SUID.txt
# find / -perm -006000 -type f –print > /tcc/arquivos_SGID.txt
```

Até agora a análise esta sendo feita com a imagem montada, ou seja, análises superficiais sem analisar a estrutura e configuração do sistema de arquivos, deixando de lado também a verificação de arquivos escondidos, excluídos ou dados “desalocados”.

Para uma análise mais completa da imagem do sistema de arquivos do sistema são necessários programas específicos para cada atividade. Para o

propósito do trabalho será utilizado um conjunto de ferramentas *open source*, chamada *The Sleuth Kit* (TSK) desenvolvida por Brian Carrier. Para a análise forense, ferramentas *open source* são úteis porque elas permitem ao perito ler o código e verificar a teoria de como a ferramenta trabalha (CARRIER, Brian: 2005). Os programas do TSK são divididos no conceito de camadas também criado por Carrier, já demonstrado na fundamentação teórica: sistema de arquivos, conteúdo, meta dados e nome de arquivos, para sistema de arquivos ext2.

Abaixo algumas das funcionalidades que podem ajudar o perito na análise da imagem.

m) Obtendo informações gerais da organização dos dados:

```
#./fsstat /tcc/imagem.dd > /tcc/fsstat.txt
```

n) Utilizando o comando *strings* é possível listar textos e procurar por palavras chaves sobre a imagem. O que proporciona uma varredura mais completa já que será o conteúdo, escondido ou não dos arquivos que serão localizados:

```
#strings /tcc/imagem.dd > /tcc/saida_strings.txt
```

o) Procurando por palavras chaves:

```
#grep –if palavras_chave.txt /tcc/saida_strings.txt > /tcc/busca_palavras_hd.txt
```

p) Listando todos os arquivos e diretórios excluídos da imagem:

```
#./fls -rd /tcc/imagem.dd > /tcc/arquivos_excluidos.txt
```

A saída dos arquivos excluídos contém o número *inode* do arquivo, no qual pode obter mais informações sobre ele ou ainda ser usado para recuperá-lo.

q) Obtendo mais informações, do arquivo com *inode* igual a 307625:

```
#./istat -f linux-ext2 /tcc/imagem.dd 307625 | less
```

r) Recuperando o conteúdo do arquivo:

```
#./icat –f linux-ext2 /tcc/imagem.dd 307625 > /tcc/arquivo_recuperado
```

Em alguns casos o número do *inode* não esta disponível, um exemplo disso é o uso do *kernel* 2.4.27 (deste caso prático). Isso significa que quando um arquivo é excluído o número do *inode* na entrada de diretório é “setado” para 0, dificultando a recuperação do arquivo. De acordo com o algoritmo de alocação de *inodes* demonstrado na fundamentação teórica o *kernel* tenta alocar primeiramente o *inode* e seus blocos de dados no mesmo grupo do diretório pai. Neste caso se o grupo do diretório pai for 19 por exemplo, pode-se gerar um arquivo com os blocos alocados para este grupo e restaurá-los com uso de ferramentas específicas.

s) Descobrindo o grupo do diretório pai da saída abaixo:

```
Saída: r/- * 0:      home/jsilva/docs/futebol-0001-800.jpg

# ./fls -Dr /tcc/imagem.dd | grep docs
++++ d/d 90361: docs
++ d/d 307625:   docs
```

t) Existem dois diretórios com o mesmo nome, para descobrir o correto basta usar o comando *find*:

```
#find /mnt/server/ -inum 307625
/mnt/server/home/jsilva/docs
```

u) Verificando o grupo no qual pertence o diretório *docs* através do *inode* encontrado:

```
# ./istat /tcc/imagem.dd 307625
inode: 307625
Allocated
Group: 75
Generation Id: 117993
uid / gid: 0 / 0
mode: dr-xr-xr-x
size: 1024
num of links: 2

Inode Times:
Accessed:          Tue Jun 13 01:24:32 2006
File Modified:     Tue Jun 13 01:24:29 2006
Inode Modified:  Tue Jun 13 01:24:29 2006

Direct Blocks:
616692
```

v) Sabendo que o grupo é 75 é possível descobrir os blocos de dados reservados para este grupo, como também o tamanho dos blocos de dados. Ambos são localizados na saída do *fsstat*:

```
CONTENT INFORMATION
--------------------------------------------
Block Range: 0 - 899607
Block Size: 1024
Reserved Blocks Before Block Groups: 1
Free Blocks: 284020

...

Group: 75:
  Inode Range: 307201 - 311296
  Block Range: 614401 - 622592
```

```
Layout:
  Data bitmap: 614401 - 614401
  Inode bitmap: 614402 - 614402
  Inode Table: 614408 - 614919
  Data Blocks: 614403 - 614407, 614920 - 622592
Free Inodes: 3671 (89%)
Free Blocks: 5953 (72%)
Total Directories: 28
```

w) O próximo passo é copiar os dados dos blocos do grupo 75, que pode ser analisado por ferramentas *data carving* como o *foremost*:

```
#dd if=/tcc/imagem.dd of=/tcc/blocos75 bs=1024 skip=614401 count=8191

# foremost -t all -i /tcc/blocos75
```

Após o uso do *foremost*, foram recuperados 10 arquivos no formato JPG, no qual não continha o arquivo procurado. O *foremost* faz uma varredura pelo cabeçalho (*header*), rodapé (*footer*) e pela estrutura de arquivos. Caso algum destes dados tenha sido sobrescritos, a captura completa do arquivo pode ser comprometida.

Outra forma importante de análise é a linha do tempo. Com a linha do tempo é possível saber a data e a hora em que um determinado arquivo foi criado, alterado ou excluído por exemplo. Neste formato o perito pode traçar de maneira mais fácil a rota de acessos aos arquivos feita pelo intruso.

x) Capturando as entradas de diretórios e logo após o meta dados (*inodes*, *timestamps*) em um mesmo arquivo:

```
#./fls -f linux-ext2 -m / -r /tcc/imagem.dd > /tcc/linhatempo_juncao.txt
#./ils -f linux-ext2 -m /tcc/imagem.dd >> /tcc/linhatempo_juncao.txt
```

y) Gerando a linha do tempo, do dia 28/05/2006 até o dia atual, ou em um intervalo de tempo definido, respectivamente:

```
#./mactime -b /tcc/linhatempo_juncao.txt -z GMT 05/28/2006 > linhatempo.txt
#./mactime -b /tcc/linhatempo_juncao.txt -z GMT 05/25/2006-05/26/2006 > linhatempo2.txt
```

8.3.2 Verificando Configurações e Estados do Sistema Operacional

8.3.2.1 Analisando Arquivos de Log

Uma análise dos arquivos de *logs* mais pertinentes se faz necessária.

8.3.2.2 Verificando Arquivos de Configurações

Uma etapa muito importante, já que através dela se pode determinar de que forma o sistema está configurado e de que forma ele se comporta. Abaixo uma lista de arquivos essenciais que devem ser analisados:

a) /etc/passwd e /etc/group: procurar por contas de usuários e grupos não autorizados ou que possui privilégios, dando maior atenção para usuários com poderes de *root*;

b) /etc/shadow: verificar se toda conta precisa de autenticação;

c) /etc/hosts: listar entradas DNS;

d) /etc/hosts.equiv: rever relacionamentos confiáveis;

e) ~/.rhosts: rever relacionamentos confiáveis baseados pelo usuário;

f) /etc/hosts.allow e /etc/hosts.deny: checar regras de permissão ou não de conexões TCP;

g) /etc/syslog.conf: determinar a localização dos arquivos de *log*;

h) /etc/rc: procurar por arquivos de inicializacao;

i) arquivos crontab: listar eventos programados;

j) /etc/inetd.conf e /etc/xinetd.conf: listar serviços ligados ao inicializar o sistema;

k) /var/log: diretório que contém os arquivos de *logs* padrão (vide /etc/syslog.conf) ;

l) /etc/security: arquivos de políticas de acesso ao sistema.

8.3.2.3 Daemons de Inicialização

A lista de processos do sistema (*daemons*) no Debian são inicializados de acordo com o nível de inicialização do sistema. Para cada um deles existe uma pasta específica (/etc/rc?d/ onde ? é o nível de inicialização) no qual possui *links* simbólicos a *shell scripts* para cada serviço na pasta /etc/init.d/. São estes *scripts* que realmente executam os binários.

Verificar quais serviços estão sendo inicializados é importante, já que o local preferido para implantar *backdoors* e *trojans* (HATCH, LEE, KURTZ: 2002). Analisar o *shell scripts* e *links* simbólicos também deve ser feita, já que pode estar redirecionado ou executando serviços diferentes do que o nome indica.

8.3.2.4 Diretório /tmp

O diretório */tmp* merece uma análise especial, pois é um diretório onde qualquer usuário pode escrever e é um repositório popular de arquivos usados por intrusos e ferramentas maliciosas (PROSISE, MANDIA: 2003). Vestígios de arquivos temporários, arquivos descompactados, *links* simbólicos, como também arquivos usados para escalação de privilégios podem ser encontrados nesta pasta.

8.3.3 Análise da Memória

Assim como na imagem do *hard disk*, o comando *strings* pode ser utilizado para carregar textos e procurar por palavras chaves. Em testes feitos, foi possível, por exemplo, encontrar partes do conteúdo de arquivo sendo visualizado e editado pelo *vi*:

```
#strings /tcc/memoria > /tcc/saida_strings.txt
```

a) Procurando por palavras chaves:

```
#grep –if palavras_chave.txt /tcc/saida_strings.txt > /tcc/busca_palavras.txt
```

O *dump* gerado através da interface */proc/kcore* pode ser analisado na busca de *rootkits* e *backdoors*. Muitas destas ferramentas manipulam as chamadas de sistemas, seja alterando a tabela de chamadas de sistemas ou fazendo um gancho (*hook*) para outras funcionalidades. Para encontrar alterações na tabela de

sistemas primeiramente é preciso resgatar a tabela de símbolos e o arquivo *vmlinux*. A tabela de símbolos (*System.map*) é gerada quando o *kernel* foi originalmente criado e reflete informações específicas sobre o *kernel* (SLOUDIS, ZELTER: 2003). Já o *vmlinux*, é um arquivo ELF que representa as estruturas do *kernel*, também gerado na compilação do *kernel*.

b) Para descobrir o endereço da tabela de chamadas de sistemas, deve ser feito uma busca na tabela de símbolos e na *vmlinux* para verificar se elas são iguais:

```
#cat System.map-2.4.27 | grep sys_call_table
c02a0dd6 R __kstrtab_sys_call_table
c02ab068 R __ksymtab_sys_call_table
c02ad794 D sys_call_table

#nm vmlinux | grep sys_call_table
c02a0dd6 R __kstrtab_sys_call_table
c02ab068 R __ksymtab_sys_call_table
c02ad794 D sys_call_table
```

c) Agora utilizando a ferramenta *gdb*, pode ser feito uma verificação de mudança na tabela de chamadas de sistemas, primeiramente listando os endereços das chamadas de sistemas originais:

```
#gdb vmlinux
(gdb) x/255 0xc02ad794

0xc02ad804 <sys_call_table+112>:    0xc0145ee0    0xc010e120    0xc013c340
        0xc0125100
0xc02ad814 <sys_call_table+128>:    0xc0125100    0xc013c5a0    0xc0116bd0
        0xc0125100
0xc02ad824 <sys_call_table+144>:    0xc013f460    0xc0124490    0xc014c300
        0xc014ad70
0xc02ad834 <sys_call_table+160>:    0xc014b150    0xc014d1e0    0xc010d9b0
        0xc0126110
0xc02ad844 <sys_call_table+176>:    0xc0125100    0xc012a470    0xc0127cc0
        0xc0128150
```

d) Listando os endereços através da memória:

```
#gdb vmlinux kcore.dd
(gdb) x/255 0xc02ad794

0xc02ad804 <sys_call_table+112>:    0xc0145ee0    0xc010e120    0xc013c340
        0xc0125100
0xc02ad814 <sys_call_table+128>:    0xc0125100    0xc013c5a0    0xc0116bd0
        0xc0125100
0xc02ad824 <sys_call_table+144>:    0xc013f460    0xc0124490    0xc014c300
        0xcc82a060
0xc02ad834 <sys_call_table+160>:    0xc014b150    0xc014d1e0    0xc010d9b0
        0xc0126110
0xc02ad844 <sys_call_table+176>:    0xc0125100    0xc012a470    0xc0127cc0
        0xc0128150
```

e) Após a listagem dos dois endereçamentos, os mesmo podem ser confrontados através do comando *diff* na detecção de discrepâncias:

```
#diff gdb_kcore1.txt gdb_kcore2.txt
10c10
< 0xc02ad824 <sys_call_table+144>:    0xc013f460    0xc0124490    0xc014c300
        0xc014ad70
---
> 0xc02ad824 <sys_call_table+144>:    0xc013f460    0xc0124490    0xc014c300
        0xcc82a060
```

Ao ser visualizado a saída completa, é possível verificar que o endereço alterado fica na posição 40. Analisando o arquivo onde ficam as definições dos endereços das chamadas de sistemas *(/usr/src/linux/ include/asm/unistd.h*) é confirmada a alteração da chamada de sistemas *mkdir*, possivelmente por alguma *backdoor*.

f) Outra forma de descobrir comprometimento no nível do kernel, através da memória é verificar o código *assembly* das chamadas de sistemas, por exemplo:

```
#gdb vmlinux
(gdb) disass sys_open
```

```
0xc013d190 <sys_open+0>:        sub    $0x14,%esp
0xc013d193 <sys_open+3>:        mov    0x18(%esp),%eax
0xc013d197 <sys_open+7>:        mov    %ebx,0xc(%esp)
0xc013d19b <sys_open+11>:       mov    %esi,0x10(%esp)
```

g) Saída junto ao dump da memória:

```
#gdb vmlinux kcore.dd
(gdb) disass sys_open
0xc013d190 <sys_open+0>:        sub    $0x14,%esp
0xc013d193 <sys_open+3>:        mov    0x18(%esp),%eax
0xc013d197 <sys_open+7>:        mov    %ebx,0xc(%esp)
0xc013d19b <sys_open+11>:       mov    %esi,0x10(%esp)
```

Comparando as duas saídas com *diff*, pode-se verificar que as mesmas não têm diferença alguma. Isso pressupõe de que não houve nenhuma mudança no código da chamada de sistema.

8.4 ANÁLISE FINAL

Após as etapas de coleta e análise das evidências, o perito forense tem como próximo objetivo averiguar e correlacionar todas as informações possíveis.

O início da investigação fica por conta da listagem dos processos:

```
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
www-data   991  0.0  4.1 16424 8016 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
www-data   992  0.0  4.2 16428 8224 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
www-data   995  0.0  4.1 16424 7944 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start -DSSL
root      1011  0.0  0.8  3004 1680 tty2     Ss   Jun12   0:00 -bash
www-data  1019  0.0  0.3  2224  740 ?        S    Jun12   0:00 nc -l -p 5600 -e /bin/bash
www-data  1020  0.0  4.1 16424 7944 ?        S    Jun12   0:00 /usr/sbin/apache2 -k start –DSSL
```

O processo com PID 1019 esta sendo executado pelo usuário *www-data*, no qual é restrito ao processo Apache, ou seja, não pode ser um usuário "logado" no sistema, já que o mesmo não possui um *shell* válido. Isso pode dizer que o serviço foi acionado por uma página do site, por exemplo.

Para conferir, foi listado os arquivos presentes na pasta padrão do site:

```
/mnt/server/var/www/apache2-default/index.php: PHP script text
/mnt/server/var/www/apache2-default/pages/adore-ng.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
/mnt/server/var/www/apache2-default/pages/configure: perl script text executable
/mnt/server/var/www/apache2-default/pages/relink: perl script text executable
/mnt/server/var/www/apache2-default/pages/relink26: perl script text executable
/mnt/server/var/www/apache2-default/pages/startadore: Bourne shell script text executable
/mnt/server/var/www/apache2-default/pages/zero.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
/mnt/server/var/www/apache2-default/pages/ava: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.0, dynamically linked (uses shared libs), not stripped
/mnt/server/var/www/apache2-default/pages/cleaner.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
/mnt/server/var/www/apache2-default/pages/symsed: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.0, dynamically linked (uses shared libs), not stripped
/mnt/server/var/www/apache2-default/exp: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.0, dynamically linked (uses shared libs), not stripped
```

Como demonstrado, existem vários binários no diretório *pages*. A verificação do usuário dono dos arquivos pode dizer a origem destes arquivos:

```
total 63
drwxr-xr-x  2 www-data www-data  1024 2006-06-13 00:58 .
drwxrwxrwx  3 root     root      2048 2006-06-13 01:00 ..
-rw-r--r--  1 root     root      9012 2006-06-13 00:56 adore-ng.o
-rwxr-xr-x  1 root     root     16463 2006-06-13 00:56 ava
-rw-r--r--  1 root     root      1020 2006-06-13 00:56 cleaner.o
-rwxr-xr-x  1 www-data www-data  4386 2006-03-16 09:53 configure
-rwxr-xr-x  1 www-data www-data   930 2004-05-26 10:50 relink
-rwxr-xr-x  1 www-data www-data  1175 2005-04-18 10:36 relink26
-rwxr-xr-x  1 www-data www-data   218 2005-04-18 10:36 startadore
-rwxr-xr-x  1 root     root     12761 2006-06-13 00:56 symsed
-rw-r--r--  1 root     root      9128 2006-06-13 00:56 zero.o
```

Analisando a lista, foi constatado de que o diretório atual (*pages*) o dono é *www-data*. O que significa que os arquivos foram baixados pelo usuário do

Apache: site ou pelo processo do *netcat*. Além disso, alguns arquivos objetos possuem como dono o usuário *root*. Verificando os arquivos excluídos, pode-se comprovar de que houve uma compilação destes arquivos com o usuário *root*.

```
r/- * 0:   var/www/apache2-default/pages/Changelog
r/- * 0:   var/www/apache2-default/pages/FEATURES
r/- * 0:   var/www/apache2-default/pages/LICENSE
r/- * 0:   var/www/apache2-default/pages/Makefile.2.6
r/- * 0:   var/www/apache2-default/pages/Makefile.2.6.gen
r/- * 0:   var/www/apache2-default/pages/README
r/- * 0:   var/www/apache2-default/pages/README.26
r/- * 0:   var/www/apache2-default/pages/adore-ng-2.6.c
r/- * 0:   var/www/apache2-default/pages/adore-ng.c
r/- * 0:   var/www/apache2-default/pages/adore-ng.h
r/- * 0:   var/www/apache2-default/pages/adore-ng.mod.c
r/- * 0:   var/www/apache2-default/pages/ava.c
r/- * 0:   var/www/apache2-default/pages/cleaner.c
r/- * 0:   var/www/apache2-default/pages/irq_vectors.h
r/- * 0:   var/www/apache2-default/pages/libinvisible.c
r/- * 0:   var/www/apache2-default/pages/libinvisible.h
r/- * 0:   var/www/apache2-default/pages/symsed.c
r/- * 0:   var/www/apache2-default/pages/visible-start.c
r/- * 0:   var/www/apache2-default/pages/Makefile
```

Mas como o intruso obteve acesso ao *root*? Analisando a lista de arquivos pode ser visualizado um arquivo ELF chamado “exp”, o que provavelmente é um *exploit* para algum serviço rodando no servidor para elevação de privilégio.

Por que depois do acesso ao *root* o intruso não excluiu o arquivo? E por que não existe um arquivo *core dump* na lista de arquivos ou mesmo nos excluídos? Provavelmente a intrusão deve ter sido feita recentemente ou até mesmo estar sendo feita naquele exato momento, então o intruso provavelmente iria excluí-lo mais tarde. Já o *core dump* não foi gerado porque o servidor esta configurado para não gerar arquivos *core dump* (core file size = 0):

```
#ulimit -a
core file size        (blocks, -c) 0
data seg size         (kbytes, -d) unlimited
file size             (blocks, -f) unlimited
max locked memory     (kbytes, -l) unlimited
max memory size       (kbytes, -m) unlimited
open files                    (-n) 1024
pipe size          (512 bytes, -p) 8
stack size            (kbytes, -s) 8192
cpu time             (seconds, -t) unlimited
max user processes            (-u) unlimited
virtual memory        (kbytes, -v) unlimited
```

Como demonstrado, a análise deve ser feita levando em conta as mais diversas fontes de evidências. O importante é o cruzamento entre elas, que pode ser traçado as ações de um intruso.

CONCLUSÃO

Perícia forense computacional é uma ciência que atualmente vem amadurecendo e aumentando sua contribuição em prol da sociedade, principalmente por utilizar meios científicos na identificação e combate aos crimes digitais. Os resultados das investigações são utilizados cada vez mais em processos judiciais, o que aumenta a necessidade de uma ótima aquisição, análise e interpretação das evidências encontradas no sistema alvo. É importante lembrar que qualquer erro neste processo pode distorcer os fatos como realmente aconteceram e interferir de forma catastrófica no resultado final de uma audiência, podendo colocar inocentes na cadeia ou soltar criminosos.

O grau de complexidade na investigação pode variar de acordo com o ambiente encontrado e principalmente pelo número elevado de aparatos tecnológicos envolvidos. O que torna a especialização uma necessidade, principalmente operados por profissionais gabaritados, utilizando técnicas, ferramentas e métodos comprovados para um bom desempenho da investigação e que esta seja bem aceita cientificamente e perante a lei.

A versatilidade e a evolução das técnicas de ferramentas utilizadas por intrusos na obtenção de acesso e permanência nos sistemas, somados à gama de funcionalidades que o Linux dispõe, obrigam o perito forense a estar sempre com seus conhecimentos atualizados, possuir raciocínio lógico, “pensar como um deles” e, dominar profundamente o ambiente a ser examinado.

As técnicas, procedimentos e ferramentas abordados durante o trabalho estão longe de serem as únicas, dependendo do ambiente em execução algumas podem ser aplicadas e outras não. De maneira geral, a prática comprovou que é possível obter e analisar em um ambiente contaminado, evidência e possíveis provas que podem ser utilizada para vários propósitos. Bem como foi comprovado que dependendo das versões e acontecimentos no servidor analisado, algumas funcionalidades encontradas livremente na literatura já não condizem com a realidade.

Para encerrar, o presente trabalho acaba abrindo portas para trabalhos futuros, os quais podem ser mais aprofundados em áreas específicas como a memória, que ainda é pouco estudada pelo seu grau de importância. Outra opção é o desenvolvimento de ferramentas que automatizem estes processos, já que dependendo da quantidade de dados podem ser cansativos. Outra vertente para pesquisa são as técnicas anti-forense que estão começando a ser utilizadas na obstrução e destruição de provas, como por exemplo: criptografia, steganografia, *data hinding*, técnicas para manipular e ludibriar valores *hash*, execução de binários remotamente, entre outros.

REFERÊNCIAS

BOVET, Daniel P., CESATI, Marco. **Understanding the Linux Kernel**. Califórnia: O'Reilly & Associates, 2003.

CARRIER, Brian. **File System Forensic Analysis**. Indiana: Addison Wesley Professional, 2005.

CASEY, Eoghan. **Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet**. 2d. Londres: Academic Press, 2004.

CHUVAKIN, Anton; PEIKARI, Cyrus. **Security Warrior**. Califórnia: O'Reilly, 2004.

FLICKENGER, Rob. **Linux Server Hacks**. Califórnia: O'Reilly & Associates, 2003.

FOSTER, James C. **Buffer Overflow Attacks: Detect, Exploit, Prevent**. Massachusetts: Syngress Publishing, 2005.

HATCH, Brian; LEE, James; KURTZ, George. **Hackers Linux Expostos**. São Paulo: Makron Books, 2002.

JESUS, Fernando de. **Perícia e Investigação de Fraude**. 2. ed. Goiania, GO: AB, 2000.

MAXWELL, Scott. **Kernel do Linux**. São Paulo: Makron Books, 2000.

MOHAY, George M. et al. **Computer and Intrusion Forensics**. Massachusetts: Artech House, Inc., 2003.

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas operacionais**. 3.ed. Porto Alegre: Sagra Luzzatto, 2004.

PROSISE, Chris; MANDIA, Kevin. **Hackers: resposta e contra-ataque**. Rio de Janeiro: Campus, 2002.

PROSISE, Chris; MANDIA, Kevin. **Incident Response & Computer Forensics**. 2d. [S.I.]: McGraw-Hill/Osborne, 2003.

RAY, John. **Maximum Linux Security**. Nova York: SAMS, 2001.

SCHWEITZER, Douglas. **Incident Response: Computer Forensics Toolkit**. Indiana: Wiley Publishing, 2003.

SLOUDIS, Ed; ZELTER, Lenny. **Malware: Fighting Malicious Code**. Nova Jersey: Prentice Hall PTR, 2003.

VACCA, John R. **Computer Forensics-Computer Crime Scene Investigation**. Massachusetts: Charles River Media, 2002.

WELSH, Math et al. **Running Linux**. 4.ed. Califórnia: O'Reilly & Associates, 2002.

ZILLI, Daniel. **Engenheiro Linux**. Rio de Janeiro: Brasport, 2003.