

Álvaro Vilobaldo Rios da Silva

Antiforense com uso de rootkits

São Paulo

2013

Álvaro Vilobaldo Rios da Silva

Antiforeense com uso de rootkits

Monografia de Conclusão de Curso apresentada à Universidade Presbiteriana Mackenzie de São Paulo como requisito parcial à obtenção do título de Especialista em Computação Forense do Curso Lato Sensu em Computação Forense

Orientadora:
Ana Cristina Azevedo

UNIVERSIDADE PRESBITERIANA MACKENZIE

São Paulo

2013

Monografia sob o título Antiforense com uso de rootkits, desenvolvida por Álvaro Villobaldo Rios da Silva, e aprovada em 28 de abril de 2013, São Paulo capital, pela banca constituída por:

Ana Cristina Azevedo
Universidade Presbiteriana Mackenzie

Ivete Irene dos Santos
Universidade Presbiteriana Mackenzie

Resumo

Analisar ambientes comprometidos por rootkits pode representar um verdadeiro desafio e junto com técnicas de antifoenses ambos podem criar mecanismos de destruição de evidências de forma sistemática e eficiente. Esta monografia é um estudo sobre rootkits e antifoense com a visão de um perito computacional forense. Baseado no funcionamento e premissas do rootkit será apresentado um processo para análise *post mortem* e as técnicas antifoenses em cada uma das suas etapas.

Palavras-chave: antifoense, rootkit, forense, evidências.

Abstract

Analyze environments compromised by rootkits can represent a real challenge and together with techniques antifoenses both can create mechanisms for destroying evidence in a systematic and efficient. This monograph is a study of rootkits and antifoense with the vision of a computer forensics expert. Based on the operation and assumptions rootkit is provided a process for *post mortem* analysis and techniques antifoenses in each of its stages.

keywords: anti-forensics, rootkit, forensic, evidence.

Lista de Figuras

1.1	Amostras Rootkit únicas Descobertas (MCAFEE, 2012)	p. 10
3.1	Processos vinculados (FLORIO, 2005)	p. 18
3.2	Processo desvinculado (FLORIO, 2005)	p. 18
4.1	Hexadecimal da Microsoft Calculator v. 5.1 visto no WinHex 16.3 SR-2 . . .	p. 24
4.2	Arquivo com MAC time original (WIKI, 2007)	p. 25
4.3	Utilização do timestomp (WIKI, 2007)	p. 26
4.4	Hello World - PIMAGE SECTION HEADER (PETRI, 2012)	p. 29
4.5	Packed Hello World - PIMAGE SECTION HEADER (PETRI, 2012)	p. 30
4.6	Registradores básicos IA32 (WISMAN, 2012)	p. 31
4.7	Registrador EFLAGS IA32 (WIKIDOT, 2010)	p. 32

Lista de Tabelas

4.1	Tipos de antifoense (BLUNDEN, 2009a).	p. 20
4.2	Probabilidade de uma colisão de hash em números sequenciais (PRESHING, 2011).	p. 27
4.3	Amostra de executáveis suportados pelo UPX (UPX, 2013).	p. 29

Sumário

1	Introdução	p. 9
1.1	Justificativa	p. 9
1.2	Objetivos	p. 10
1.2.1	Objetivo Geral	p. 10
1.2.2	Objetivo Específico	p. 10
1.3	Descrição dos Capítulos	p. 11
2	Ciência Forense Computacional	p. 12
2.1	Definição	p. 12
2.2	Competências do Perito no Brasil	p. 13
3	Rootkit	p. 14
3.1	Malware	p. 14
3.2	Rootkit	p. 15
3.3	Tipos de Rootkit	p. 15
3.4	Controle Sobre Execução	p. 16
3.5	Manipular Dados do Kernel	p. 17
3.6	Considerações	p. 19
4	Análise de Rootkits e Técnicas Antiforense	p. 20
4.1	Análise de Rootkits	p. 20
4.2	Cópia forense do HDD	p. 22
4.3	Recuperar arquivos	p. 23

4.4	Coletar metadados dos arquivos	p. 25
4.5	Retirar arquivos conhecidos	p. 26
4.6	Análise estática dos executáveis suspeitos desconhecidos	p. 27
4.6.1	Packer	p. 28
4.6.2	Bytecode	p. 30
4.7	Análise dinâmica dos executáveis suspeitos	p. 31
5	Conclusão	p. 33
	Referências Bibliográficas	p. 34

1 Introdução

O rootkit é uma ferramenta utilizada em geral por atacantes avançados com fins maliciosos e oferece grande obstáculo para ser detectado justamente por ser furtivo e sofisticado recurso de controle de sistemas operacionais. Seu uso força o perito a ter um conhecimento amplo e usar técnicas também sofisticadas para investigá-lo. Logo entender o seu funcionamento é determinante para uma perícia bem sucedida.

Esta monografia apresenta uma incursão na antifoense focada na utilização de rootkits em ambientes *Microsoft Windows*, visando mostrar o que é um rootkit e os principais empecilhos que o perito pode encontrar ao analisá-lo. Para desenvolver esse tema foi utilizada vasta bibliografia e referências reais para melhor entendimento das técnicas de antifoense computacionais utilizadas ou idealizadas que podem ser utilizados por rootkits.

1.1 Justificativa

Conforme o mundo digitalizou-se, digitalizaram-se também as suas ameaças, onde antes se podia ver mesmo que por instantes mísseis ou bombas sendo lançada hoje temos inúmeras ameaças invisíveis que podem causar tanto estrago quanto, contudo na luz do princípio de Locard que diz que tudo que é tocado deixa vestígios, talvez essas ameaças não sejam tão invisíveis. O perito deve estar preparado para a ação de um usuário avançado que conheça bem o sistema operacional atacado, comprometido ou usado. Mesmo que não seja algo corriqueiro na rotina da grande maioria dos profissionais, encontrar um atacante de alto nível trará novos desafios e obstáculos tão pouco corriqueiros quanto. Saber como dificultar ou impossibilitar o trabalho do perito é como ele poderá evitar a armadilha de achar que no corpo (corpo de delito) investigado não existe nada.

A figura 1.1 mostra que quantidade de amostras únicas encontradas de rootkits entre 2009 e 2012 por trimestre nunca foi abaixo de 100 mil. Esses dados indicam que rootkits são ameaças reais e constantes. Estudar seu comportamento e entender como ele pode ser usado na antifo-

rense, permite ao perito lidar corretamente com rootkits.

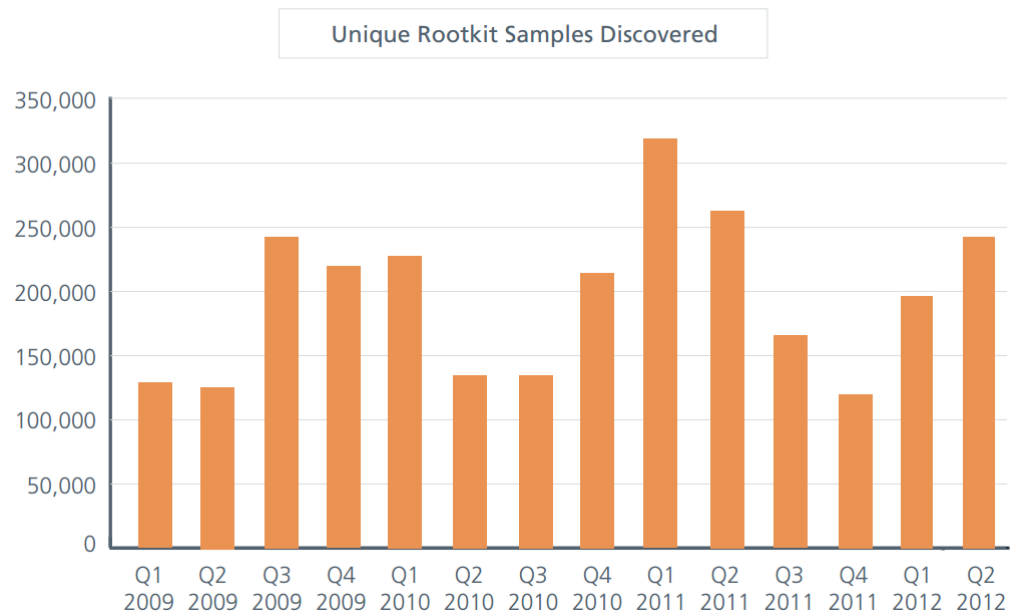


Figura 1.1: Amostras Rootkit únicas Descobertas (MCAFEE, 2012)

Com um processo bem definido o perito tende a diminuir o tempo de análise e um melhor aproveitamento das mesmas. Sendo assim, esse trabalho poderá ajudar a traçar um processo bem elaborado de trabalho que seja rápida e eficiente sem deixar brechas que permitam ou ajudem ações antifoenses provendo mais qualidade com mais precisão.

1.2 Objetivos

Esse estudo pode ajudar a esclarecer diversos tópicos obscuros a respeito do que pode ser encontrado em investigações forenses computacionais quando o perito se vê diante de ameaças persistentes e elaboradas como um rootkit e consequentemente pode ajudar a resolver casos onde foram feitas antifoense.

1.2.1 Objetivo Geral

Apresentar um processo para análise *post mortem* de computadores com Microsoft Windows e as principais técnicas antifoense que podem ser encontradas em cada fase do processo.

1.2.2 Objetivo Específico

Segue os objetivos específicos utilizados para alcançar o objetivo principal:

- Apresentar o que é ciência forense, para que serve a ciência forense e a legislação que garante a existência do perito no Brasil;
- Expor o que é um rootkit, seu funcionamento e seus tipos;
- Mostrar o que é antiforense computacional categorizando seus tipos; e
- Apresentar um processo de análise de rootkit.

1.3 Descrição dos Capítulos

Ao longo dos capítulos são desenvolvidos os conhecimentos para o entendimento do tema proposto, onde cada um foi dividido em um assunto específico.

No capítulo *Ciência Forense Computacional*, será mostrado uma visão geral do que é a ciência forense, como ela é aplicada na computação, o que é um perito forense computacional e o respaldo legal para sua atuação.

No capítulo *Rootkit*, conduz a uma visão do funcionamento de um rootkit, algumas técnicas para subverter os sistema, o que é um *malware*, a origem do nome rootkit, o que significa ser o usuário *root*, os objetivos do rootkit, suas premissas e sua definição. Também será mostrado os tipos de rootkits e analisado duas categorias gerais que exemplificam como o rootkit pode se manter oculto.

Por fim no último capítulo *Análise de Rootkits*, será apresentado um processo de análise *post mortem* com passos que irão mostrar em cada etapa o que deve ser feito, técnicas antiforense e o quão complicado e impactante ela é.

2 *Ciência Forense Computacional*

A humanidade sempre teve assuntos divergentes, onde em outrora esses assuntos eram resolvidos com o mais forte ou hauto sobrepujando a vontade doutro, agora com o advento da civilização se convencionou o uso de um mediador neutro. Quando trata-se de assuntos de disputa de interesse, espera-se que o mediador busque e chegue o mais próximo possível da verdade antes de tomar uma decisão.

O juiz é um exemplo claro do que é um mediador, contudo em diversos conflitos se faz necessário conhecimentos técnicos específicos. Nesses casos ele é auxiliado por um especialista ou perito no assunto técnico em discussão. Para tal, esse especialista usa a ciência forense para trazer luz os fatos respaldando a decisão do juiz sobre o assunto.

2.1 Definição

Segundo Houaiss e Villar (2009) ciência é um “corpo de conhecimentos sistematizados adquiridos via observação, identificação, pesquisa e explicação de determinadas categorias de fenômenos e fatos, e formulados metódica e racionalmente.” e forense é “relativo aos tribunais e à justiça”. Logo as ciências forenses é a utilização da ciência “à análise de vestígios, no intuito de responder às demandas judiciais” (VELHO; GEISER; ESPINDURA, 2012, p. 3).

De certo a área médica foi a primeira a ser requisitada em tribunais construindo técnicas que levaram ao desenvolvimento ao longo dos anos da medicina legal. No Império Romano médicos eram chamados para lucidar mortes diz França (2008), outro exemplo histórico da importância do parecer técnico pode ser visto no *Código Criminal Carolino* feito em 1532 por Carlos Magno que definia a análise médica em determinados crimes.

Com o avanço da tecnologia e o valor da perspectiva especializada, foi natural que outras áreas também fossem utilizadas no foro. A computação forense, apesar da tecnologia e inovação inatas, conceitualmente ainda se propõe a grosso modo a usar a ciência para mostrar fatos demandados judicialmente, como podemos ver em Machado e Eleutério (2011), “[...]”

computação forense tem como objetivo principal determinar a dinâmica, a materialidade e autoria de ilícitos ligados à área da informática, tendo como questão principal a identificação e o processamento de evidências digitais em provas materiais [...], por métodos técnicos-científicos, conferindo-lhes validade probatória em juízo.”.

2.2 Competências do Perito no Brasil

No código de processo penal temos o perito sendo tratado no *capítulo II: do exame do corpo de delito, e das perícias em geral*. O artigo 158 do Código de Processo Penal (CPP) indica que quando o crime deixa vestígio deve ser realizado o corpo de delito, ou seja, no artigo 158 é definido quando o perito se faz necessário.

Art. 158. Quando a infração deixar vestígios, será indispensável o exame de corpo de delito, direto ou indireto, não podendo supri-lo a confissão do acusado. (BRASIL, 1941)

No artigo seguinte é descrito segundo o CPP, quem pode ser o perito.

Art. 159. O exame de corpo de delito e outras perícias serão realizados por perito oficial, portador de diploma de curso superior. (Redação dada pela Lei nº11.690, de 2008)

§1º Na falta de perito oficial, o exame será realizado por 2 (duas) pessoas idôneas, portadoras de diploma de curso superior preferencialmente na área específica, dentre as que tiverem habilitação técnica relacionada com a natureza do exame. (Redação dada pela Lei nº11.690, de 2008) (BRASIL, 1941)

Caso não haja perito oficial o juiz deve indicar alguém que não pode recusar sobre pena de multa, a menos que tenham algo que os desqualifique como tal, como ser parente de uma das partes, ter interesse na causa etc. Mesmo não sendo oficiais esses peritos estão sujeitos aos deveres, direitos e punições dadas os peritos oficiais.

Já no código de processo civil (CPC), temos o perito sendo mencionado na *seção VII: Da Prova Pericial* e define de forma bem semelhante quando um perito deve ser indicado. No artigo 420 do CPC temos a definição da prova pericial e quando ela é descartada.

Art. 420. A prova pericial consiste em exame, vistoria ou avaliação.
Parágrafo único. O juiz indeferirá a perícia quando:

- I - a prova do fato não depender do conhecimento especial de técnico;
- II - for desnecessária em vista de outras provas produzidas;
- III - a verificação for impraticável.

(BRASIL, 1973)

A diferença no caso do CPC é que o perito não precisa ser oficial, basta o juiz indicá-lo.

3 *Rootkit*

Ambientes informatizados trazem novas ameaças cada uma com suas próprias características, também é comum a combinação dessas características para formar novas ameaças. Nessa parte serão mostradas as definições básicas de algumas das ameaças mais comuns e a diferença entre elas.

É importante ressaltar que existem aplicações legítimas e legais no uso do rootkit como em computadores corporativos e aplicações investigativas, contudo é inerente do rootkit obter acesso *root* e manter-se oculto. Dessa forma ele age contra o usuário do sistema e por esse motivo ele vai ser tratado como *malware*.

3.1 **Malware**

Malware é a junção das palavras em inglês *malicious* com *software*, esses termos em inglês pode ser traduzido livremente como aplicativo ou código malicioso. Dessa forma podemos considerar como *malware* os vírus, *worms*, *trojans*, *botnets* e outros, contudo o inverso não é válido, ou seja, todo vírus é um *malware*, mas nem todo *malware* é um vírus.

Os vírus e *worms* são feitos para espalharem-se, a diferença está em como eles se espalham. O vírus precisa ser ativado ou executado pelo usuário e em geral fica atrelado a algum executável que pode ser ou não um programa legítimo subvertido (LUDWIG, 1995), ao contrário do *worm* que não precisam da ação direta do usuário para se espalhar e permanece apenas na memória (MCAFEE, 2013).

Uma *botnet* é uma rede de computadores controlados por cibercriminosos (KASPERSKY, 2013). Basicamente quando um computador é infectado pelo agente da *botnet* o mesmo se torna parte da rede e é controlado pelo dono da *botnet*.

3.2 Rootkit

No universo *UNIX*¹ ou *UNIX-like*² a conta de usuário com menor restrição de segurança é referenciada como conta *root* sendo que em alguns sistemas o nome de usuário é literalmente *root*, mas isso é apenas uma convenção histórica e não uma imposição (BLUNDEN, 2009b). Enquanto *kit* significa conjunto de peças (SANTOS, 2000).

Os primeiros rootkits apareceram a cerca de 20 anos já no fim dos anos 80 e início dos 90, quando foram percebidos comportamentos anormais em computadores como espaço em disco utilizado sem identificação, conexões de rede não-listadas e uso anormal do CPU (ROSANES, 2011). Rootkit pode ser visto como um *kit* composto de pequenos programas úteis, por exemplo, binários, *scripts*, arquivos de configuração que permitem a um atacante manter o acesso “root”. Em outras palavras, um rootkit é um conjunto de programas e de códigos, que permite a presença permanente ou consistente, não detectável em um computador (HOGLUND; BUTLER, 2005).

Essa coleção de ferramentas que permitem aos invasores ocultarem suas atividades em um computador, de modo que eles podem secretamente monitorar e controlar o sistema por um período prolongado. Existem três serviços que são comuns aos rootkits: ocultação, comando e controle (C2) e vigilância (BLUNDEN, 2009b). Ocultação, o rootkit deve passar despercebido, sem que o usuário e/ou antivírus o detecte; vigilância ou monitoramento basicamente consiste em acompanhar as ações do usuário, o rootkit tem que ser capaz de saber o que o usuário está fazendo; e comando e controle, permite ao dono do rootkit o controle remoto sobre o mesmo, definindo suas ações e direcionando-as. Desses três serviços o mais importante e obrigatório para o rootkit é a furtividade, pois um rootkit detectável vai durar muito pouco.

Como pode ser visto cada um desses agentes subversivos tem definições e características próprias diferentes e uma em comum todos eles subvertem o sistema de alguma forma.

3.3 Tipos de Rootkit

Para Medina (2009) os rootkits são divididos em quatro grupos:

- Ring 3 (*user-mode*): são rootkit que operam acima do sistema operacional como um aplicativo, mudando um aplicativo padrão ou injetando código na memória na área reservada a aplicações *user-mode*;

¹Um sistema operacional multiusuário amplamente utilizado.

²Sistemas Operacionais baseados no Unix, como o GNU/Linux

- Ring 0 (*kernel-mode*): esse tipo de rootkit roda no mesmo nível do *kernel* e operam injetando código na área da memória reservada ao *kernel*, adicionando código ao *kernel* ou modificando *drivers*;
- Hardware/Firmware: os rootkit se instalam em *firmwares* como os *firmwares* das de rede ou placas de vídeo, de forma que rodam independentes do sistema operacional; e
- Virtualização: feitos para executarem em ambientes virtualizados, estes rootkits executam junto ao sistema de virtualização para interceptar ou controlar as *virtual machines*.

Como pode ser notado, o tipo de ambiente onde o rootkit executa define o seu tipo, por exemplo, um rootkit que só executa em memória, será um *user-mode* ou *kernel-mode* rootkit dependendo do lugar na memória que o mesmo irá utilizar.

Manter o controle sobre o usuário permite ao rootkit continuar escondido. Butler (2006) mostra em seu trabalho técnicas para manter o rootkit oculto, para isso o mesmo deve ganhar controle sobre a execução e/ou manipular o dados do *kernel*.

3.4 Controle Sobre Execução

Para ganhar controle sobre a execução do sistema o rootkit pode substituir programas do sistema operacional por uma versão que faça o mesmo, mas que ignore a ação do rootkit ou redirecionar o fluxo de uma chamada *hook*.

Windows application programming interface (Windows API) é uma interface da família Microsoft Windows descrita no *Windows Software Development Kit (SDK)* que podem ser chamadas em *user-mode* (MICROSOFT, 2008). A *Windows API* nada mais é do que um conjunto de funções expostas para uso de aplicativos em geral, por exemplo, quando um aplicativo feito para Microsoft Windows precisa fazer algo abstraído pelo sistema operacional como criar arquivos, listar processos etc, o mesmo provavelmente irá utilizar essas *functions* para tal tarefa. Apesar cada linguagem de programação possui métodos ou *functions* próprias para criar um arquivo, cada qual com sua sintaxe e peculiaridades, contudo é bem provável que sejam apenas outra abstração e que na realidade o mesmo use a *function CreateFile* disponibilizada pelo *Windows API* (RUSSINOVICH; SOLOMON; IONESCU, 2012).

Native system services ou *system calls* são serviços do sistema operacional não documentados que são disponibilizados para uso em *user-mode*, sendo inclusive chamados por *functions* do *Windows API*, por exemplo, *NtCreateUserProcess* é o serviço interno do sistema que a

function CreateProcess do *Windows API* chama para criar novos processos. Já as chamadas *Kernel support functions* ou *routines* também são sub-rotinas do sistema operacional Microsoft Windows, contudo essas são apenas chamadas em *kernel-mode* (RUSSINOVICH; SOLOMON; IONESCU, 2012, p. 4).

Ao criar um *breakpoint* o processador redirecione o fluxo de determinado programa para outro para que o mesmo possa depurar (MICROSOFT, 2013a). Desse jeito ao incluir um *breakpoint* em alguma *function* do *Windows API*, algum serviço do *system calls* ou ainda na chamada de uma *routine* o rootkit recebe o controle da execução desses processo como se fosse um *debugger*. Como foi explicado anteriormente algumas dessas funções e/ou chamadas podem ser utilizadas tanto por rootkits que trabalham em *user-mode* quanto pelos que trabalham em *kernel-mode*.

Outra forma de *hook* é interceptar chamadas a tabela de interrupções básicas. Basicamente o processador pode receber dois tipos de exceções, uma gerada por hardware chamada de externa e interrupções geradas por programas (INTEL, 2011, p. 6-2 Vol. 3A). Essas interrupções podem variar, desde uma tecla sendo apertada pelo usuário, mudança de contexto do processador, ou até um *chipset* da placa mãe informando que a transferência entre o disco e a memória foi completado (DMA).

A grosso modo, quando uma interrupção ou exceção ocorre o processador usa o endereço de memória armazenado no índice correspondente a interrupção lançada que por definição aponta para uma *procedure* que trata a interrupção (INTEL, 1986), ou seja, esse endereço na memória indica qual é a *procedure* específica para a interrupção ou exceção que ocorreu. O rootkit pode interceptar uma chamada a tabela estrutura de dados com índices dessas tratativas como *Interrupt Vector (IVT)* em modo real ou a *Interrupt Descriptor Table (IDT)* em modo protegido, ambas com estruturas e funções similares, de forma que mude o fluxo de tratativa.

3.5 Manipular Dados do Kernel

Outra forma do rootkit se manter oculto é manipular dados providos da estrutura do *kernel*, como mudar estruturas do mesmo desvinculando processo e *drivers* de listagens de processos ou ainda alterando tabelas de *handle* (BUTLER, 2006, p. 16) esse que é o caso do rootkit para Microsoft Windows XP FU rootkit criado pelo pesquisador Jamie Butler, que manipula diretamente objetos do *kernel* sem o uso de *hook*, para isso ele muda os dados dentro da estrutura do *EPROCESS* (FLORIO, 2005, p. 4).

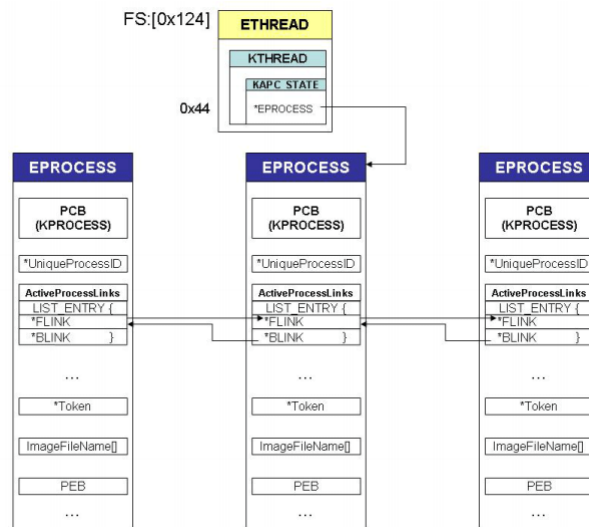


Figura 3.1: Processos vinculados (FLORIO, 2005)

No sistema operacional Microsoft Windows todos os processos possuem uma estrutura *EPROCESS* que é vinculada a um processo prévio e o um posterior, como pode ser visto na figura 3.1 formando uma lista de processo (*double-linked list*). O que o rootkits como o FU rootkit, também chamados de *Direct Kernel Object Manipulation (DKOM)* fazem é desvincular o processo desse lista deixando como na figura 3.2 (FLORIO, 2005).

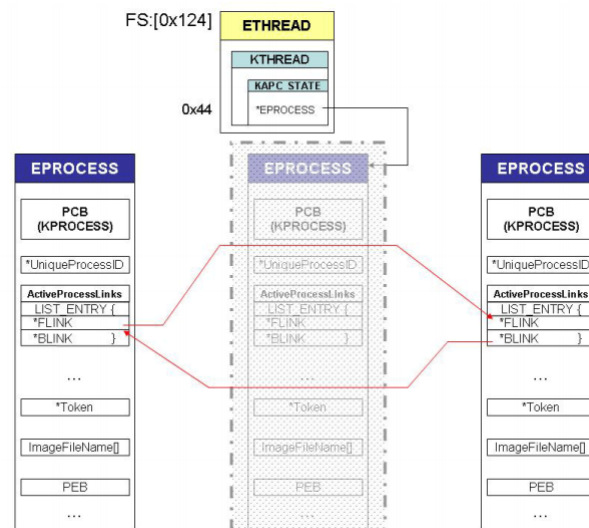


Figura 3.2: Processo desvinculado (FLORIO, 2005)

3.6 Considerações

Como foi mostrado de algum modo os rootkits sempre tentarão esconder sua presença e seus rastros, logo a antiforeshense faz é parte do seu conceito. Hoglund e Butler (2005) dizem que rootkits existem pelo mesmo motivo que grampos telefônicos ou espiões, existem para que pessoas possam saber ou controlar outras. Apesar de em teoria esses métodos serem facilmente compreendidos, na prática em um sistema operacional moderno como o Microsoft Windows 7 que possui inúmeras proteções, usar qualquer um desses métodos exige muita pesquisa. Por essa razão o desenvolvimento de rootkits exige conhecimentos profundos tanto de arquitetura de computadores quanto do funcionamento do sistema operacional tornando-o muito valioso.

4 *Análise de Rootkits e Técnicas Antiforenses*

Harris (2006) define antiforenses como método para prevenir ou agir contra a ciência usada a favor das leis civis e criminais que são aplicadas por órgãos como a polícia. Berinato (2007), amplia essa ideia mostrando que é mais que uma técnica usada, é uma abordagem criminosa. Assim todas as tentativas de interferir na existência, quantidade e/ou qualidade da evidência de uma cena de crime, ou mesmo fazer a análise e exame das provas difíceis ou impossíveis de realizar, para Rogers (2005) será considerada antiforenses. Dessa forma concluímos resumidamente que a antiforenses computacional pode ser definida como qualquer ação praticada para obstruir, dificultar ou destruir evidências ou provas no âmbito computacional.

Dentre as modalidades de antiforenses computacional, Blunden (2009a) as categoriza em cinco grandes grupos, veja a tabela 4.3:

Tipo	Descrição
Destruição de dados	Destruir arquivos ou metadados
Ocultação de dados	Salvar arquivos em lugares incomuns ou não convencionais
Corrupção de dados	Compactação, criptografia ou mudar o seu <i>file format</i>
Fabricação de dados	Introduzir <i>known files</i> ou pistas falsas
Não uso de disco	Contracepção de dados ou injeção em memória

Tabela 4.1: Tipos de antiforenses (BLUNDEN, 2009a).

Esses tipos serão apresentados durante as fases da análise *post mortem*.

4.1 *Análise de Rootkits*

Na ciência forense computacional o corpo de delito quase sempre consiste em computadores e quando o perito vai realizar a coleta do material para a análise ele pode encontrar esse dispositivo ligado ou desligado. Quando o mesmo está ligado é possível realizar a análise viva, entretanto a análise *live* de um ambiente com rootkit pode gerar dados falsos ou imprecisos.

Análise *live* ou análise viva, consiste em uma análise focada em extrair e examinar dados voláteis (MCDOUGAL, 2006), esses dados voláteis são perdidos ao desligar o computador, como por exemplo, o conteúdo de registradores do processador, dados na *cache*, dados na memória etc. Essa abordagem possui diversas vantagens entre elas estão extração de dados voláteis como já citado, triagem de equipamentos, triagem de dados, preservação de dados criptografados e a possibilidade de estabelecer flagrante (MESQUITA; HOELS; RALHA, 2011).

É importante ressaltar que toda ferramenta ou *hardware* utilizado para a coleta de dados em equipamentos que ainda estão em execução, vai depender de dados fornecidos pelo equipamento periciado, ou seja, em algum momento essas ferramentas forenses vão requisitar dados de um sistema comprometido e passível de alguma interceptação. Rutkowska (2007) mostra que mesmo quando a solução utilizada para coleta é um hardware isso pode acontecer. Logo em um ambiente dominado por um rootkit não é um ambiente confiável, pois ele pode interceptar e alterar chamadas das ferramentas de análise, impossibilitando uma coleta real e fiel.

A metodologia empregada na análise de um ambiente supostamente com um rootkit varia muito, mas em termos gerais a análise *post mortem* é menos arriscada. As medidas antiforense que podem ser tomadas pelo rootkit dependem do ambiente de execução e engloba também a equipe que o administra.

O melhor dos cenários para o atacante é quando o sistema é administrado por leigos, não capacitados, sobrecarregados e/ou os sistemas não são atualizados com frequência tornando mais fácil para o rootkit esconder-se. No pior dos cenários o administradores são altamente qualificados, realizam rotinas frequentes e o ambiente esta sempre atualizado e auditorado (BLUNDEN, 2009a). Entender isso pode indicar ao perito o quão complexo o rootkit é, pois quanto mais próximo do pior cenário mais avançado tem que ser o rootkit o que pode caracterizar um *Advanced persistent threat (APT)* ou ameaça avançada e persistente, ou seja, o ataque utilizando o rootkit foi orquestrado e planejado apenas para esse alvo específico.

Não existe técnica 100% antiforense, então o objetivo do atacante é tornar a análise cansativa para induzir ao erro podendo adicionar inclusive falsos rootkits ou *malwares* para despistar, iludir ou levar a conclusões precipitadas. Em (BLUNDEN, 2009a) são mostrados passos para a análise *Post mortem* onde se suspeita de rootkit:

- Clone do *hard disk drive (HDD)*;
- Recuperar arquivos;
- Coletar metadados dos arquivos;

- Retirar arquivos conhecidos;
- Análise estática dos executáveis suspeitos desconhecidos; e
- Análise dinâmica dos executáveis suspeitos.

Esses passos pode ser vistos como filtros, onde cada passo retira cada vez mais ruído, eliminando em cada etapa arquivos supérfluos. A seguir serão descritos cada um dos passos e as técnicas antifoenses que o perito pode enfrentar.

4.2 Cópia forense do HDD

A primeira fase e mais importante é a coleta, que nesse caso é clone do HDD. O clone do HDD, pode ser feito com software, como o DD¹ ou por hardware específico para clonagem de HDDs. Por via de regra essas ferramentas são homologados por órgãos como *National Institute of Standards and Technology (NIST)* que atestam a qualidade e eficácia dessas ferramentas em produzir cópias fidedignas que podem ser usadas em disputas legais como evidências.

Basicamente todos essas ferramentas fazem cópias de setor por setor de todo o disco, para que o perito possa trabalhar nessa cópia preservando o HDD original. Essas cópias já foram exaustivamente usadas em tribunais e são muito úteis e indispensáveis para a análise do perito. Hoje sua eficácia não é questionada, pois essa cópia ou clone tem os dados idênticos ao do HDD objeto de análise e essa igualdade pode ser atestada com *checksums* como os algoritmo de *hash MD5* ou *SHA1* (MULVEY, 2007). Essa abordagem pode ter alguns empecilhos e o rootkit pode não vir a ser clonado junto com os dados do HDD.

Primeiro e mais óbvio é que o rootkit pode nem estar no disco clonado, como estar somente na memória ou ser alocado em um outro dispositivo. O rootkit poder ser projetado para ficar somente na memória, contudo isso pode impedir que o mesmo persista já que é apagado sempre que o computador desliga (MEDINA, 2009).

Durante o *boot* de um *Personal Computer (PC)* moderno, o *firmware* de *boot* inicia o *Basic Input/Output System (BIOS)* dando suporte básico para os principais periféricos como teclado e vídeo em modo texto. Em seguida é iniciado o *Power-On Self-Test (POST)* que além de realizar alguns testes inicia com base das configurações do *SETUP* salvas na memória CMOS todos os circuitos periféricos, vídeo em modo gráfico e passa o controle ao sistema operacional (TORRES, 2001).

¹<http://pubs.opengroup.org/onlinepubs/9699919799/utilities/dd.html>

Durante todo esse processo o computador é executado em *Real Mode*, sendo que em modo real os programas não são isolados uns dos outros como no *Protected Mode*. Logo qualquer programa em execução pode acessar todo conteúdo na memória. O rootkit pode persistir no *firmware*² de algum periférico, como a placa de vídeo (ECONOMOU; JUAREZ, 2012) ou placa de rede (BLANCO; EISSLER, 2012), ou seja, ele irá executar antes do sistema operacional com todos os benefícios do modo real e tornar o clone do HDD inútil.

O perito deve ter em mente ainda que o rootkit pode estar em áreas reservadas do disco como *host protected area (HPA)* ou *Device configuration overlay (DCO)*, essas áreas são desenvolvidas para não serem modificadas ou mesmo acessadas por usuários, BIOS ou sistema operacional (GUPTA et al., 2006). Dessa forma dependendo da ferramenta utilizada para clone do HDD pode ser que essas áreas não sejam clonadas ((NIST, 2008a) e (NIST, 2008b)), inutilizando o clone do HDD.

Outro assunto que o perito deve se ater é ao que o disco clonado pode ter *full disk encryption* o que significa que todos os dados no disco estão criptografados (RUBENS, 2012) e que o clone também terá todos os dados criptografados. Logo o disco deverá ser descriptografado antes da análise. Entretanto sendo improvável que um rootkit criptografe todo o disco da vítima, pois chamaria muita atenção e iria de encontro a uma de suas premissas que é se ocultar no sistema. Então quando disco está criptografado, ele provavelmente foi feito pela vítima e conseqüentemente a maior interessada no trabalho do perito facilitando a descriptografia do disco.

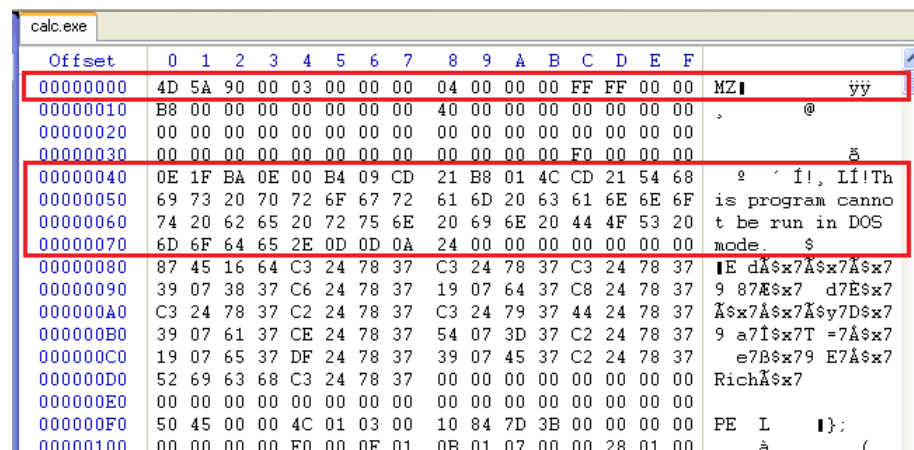
4.3 Recuperar arquivos

O próximo passo a ser tomado deve ser recuperar todos os arquivos do disco, isso inclui todos os arquivos *Master Table File (MFT)* deletados ou não, seguido por *data carving* com especial preocupação com:

- Assinaturas de executáveis;
- Fragmentos de arquivos;
- Data streams alternativos; e
- Slack space;

²Todo programa salvo numa *Read-only memory (ROM)* é chamado de *firmware*

Todos os arquivos tem um especificação que define o formato que o arquivo deve ter para ser corretamente interpretado, essas especificação define um padrão de como o *bits* são organizados internamente e é chamado de *file format* (ROUSE, 2005). Esse formato também é usado por ferramentas de *data carving* para encontrar arquivos. Por exemplo a especificação do *Portable Executable (PE) File Format* (MICROSOFT, 2013b) determina que todos os programas comecem com um pequeno executável *MS-DOS*. Esse pequeno executável inicia com o *Magic Number* 0x5A4D ou *MZ* em *ASCII* (iniciais de Mark Zbikowski um dos arquitetos originais do MS-DOS (PIETREK, 2002)), seguido de uma mensagem como pode ser vista na figura 4.1. A função desse pequeno programa é apresentar a mensagem “This program cannot be run in DOS mode”.



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	@
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	F0	00	00	
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	?
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	program
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	cannot
00000080	87	45	16	64	C3	24	78	37	C3	24	78	37	C3	24	78	37	be
00000090	39	07	38	37	C6	24	78	37	19	07	64	37	C8	24	78	37	run
000000A0	C3	24	78	37	C2	24	78	37	C3	24	79	37	44	24	78	37	in
000000B0	39	07	61	37	CE	24	78	37	54	07	3D	37	C2	24	78	37	DOS
000000C0	19	07	65	37	DF	24	78	37	39	07	45	37	C2	24	78	37	mode
000000D0	52	69	63	68	C3	24	78	37	00	00	00	00	00	00	00	00	.
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000F0	50	45	00	00	4C	01	03	00	10	84	7D	3B	00	00	00	00	
00000100	00	00	00	00	F0	00	0F	01	0B	01	07	00	00	28	01	00	

Figura 4.1: Hexadecimal da Microsoft Calculator v. 5.1 visto no WinHex 16.3 SR-2

Programas que fazem *data carving* buscam essas estruturas definidas para encontrar arquivos, mas o rootkit pode usar programas como o *Transmogrify* do *Metasploit Anti-Forensics Project (MAFIA)* que é capaz de mascarar um arquivo em qualquer assinatura (MAYNOR; MOOKHEY, 2007), de forma a deixar um executável no formato de um arquivo texto por exemplo.

Outra forma do rootkit esconde-se de um *data carving* é utilizando-se de uma organização de arquivo não convencional, como usar fragmentos de diversos arquivos que só fazem sentido quando organizados de determinada forma, salvar seus arquivos em *slack space* buscando e indexando-os em um *file system* próprio (EASTTOM, 2007), ou ainda usando *features* de um sistema de arquivos como as *streams* do NTFS (THOMPSON; MOROE, 2006). Entretanto por mais eficiente que seja o método utilizado, o rootkit em algum momento tem que executar, logo em algum ponto o rootkit tem que trabalhar com estruturas convencionais para que o sistema operacional o interprete, mesmo que seja apenas um *stub* (ver análise estática).

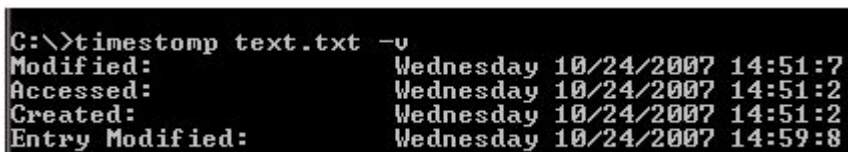
4.4 Coletar metadados dos arquivos

Nessa etapa é feita a coleta dos metadados de todos os arquivos, segue os principais:

- Hash;
- MAC time;
- Nome;
- Localização completa do arquivo em disco (*Path*); e
- Tamanho;

É importante que se tenha ao menos esses metadados dos arquivos recuperados. Os próximos passos podem depender de um desses elementos. O *hash* é fundamental para o próximo passo, o tamanho vai ser abordado na análise estática, o nome e *path* são obviamente para sua identificação e para localização, e o *MAC (Modify, Access, Create) time* pode vir a ser utilizado para desenvolver uma *timeline*, ferramenta útil e poderosa para favor do perito.

A análise da *timeline* é um importante passo para todo processo tradicional de investigação, com base nela um perito forense computacional pode extrair informações cruciais para o caso (GUOJONSSON, 2010). Criar uma *timeline* de eventos com base nos *MAC time* (figura 4.2) dos arquivos ajuda a entender o ciclo de contaminação, além de levantar suspeitas sobre vários dados inconsistentes, como por exemplo, porque o papel de parede do Windows tem a data de modificação ou criação recente? Além de ajudar a entender como o rootkit pode ter se mantido no ambiente analisado. Dessa forma é possível recriar a cadeia de eventos gerando ao mesmo tempo contexto.



```
C:\>timestomp text.txt -v
Modified:      Wednesday 10/24/2007 14:51:7
Accessed:      Wednesday 10/24/2007 14:51:2
Created:       Wednesday 10/24/2007 14:51:2
Entry Modified: Wednesday 10/24/2007 14:59:8
```

Figura 4.2: Arquivo com MAC time original (WIKI, 2007)

Para Hallman (2011) um dos principais desafios na análise de *timeline* envolve a fácil manipulação. Para não levantar suspeitas, ou dificultar o processo de análise da *timeline*, o rootkit pode ser programado a alterar o *MAC time* quando necessário. O programa *Timestomp* também do *MAFIA* e hoje integrado ao *Meterpreter* é capaz de alterar esses metadados como é mostrado na figura 4.3.

```

C:\>timestamp text.txt -m "Monday 1/01/2001 01:01:1 AM"

C:\>dir
Volume in drive C has no label.
Volume Serial Number is 3036-18D7

Directory of C:\

05/26/2007  06:01 PM                0 AUTOEXEC.BAT
05/26/2007  06:01 PM                0 CONFIG.SYS
10/24/2007  02:58 PM               <DIR>      Documents and Settings
05/29/2007  01:15 PM               <DIR>      Program Files
01/01/2001  01:01 AM                0 text.txt
10/24/2007  02:50 PM          57,344 timestamp.exe
06/18/2007  05:31 PM               <DIR>      WINDOWS
               4 File(s)          57,344 bytes
               3 Dir(s)  11,767,320,576 bytes free

C:\>timestamp text.txt -v
Modified:                Monday 1/1/2001 1:1:1
Accessed:                Wednesday 10/24/2007 14:51:2
Created:                 Wednesday 10/24/2007 14:51:2
Entry Modified:          Wednesday 10/24/2007 14:59:8

C:\>

```

Figura 4.3: Utilização do timestamp (WIKI, 2007)

4.5 Retirar arquivos conhecidos

No passo anterior foram retirados *hashes* dos arquivos encontrados e eles podem ser comparados com bases de *hashes* ou *hashset* conhecidos, como os fornecidos pelo *National Software Reference Library (NSRL)* ³ fornecidos pelo NIST. Comparando os *hashes* dos arquivos com *hashset* de arquivos conhecidos é possível retirar todos os arquivos conhecidos maliciosos também conhecidos como *known bad* ou arquivos inequivocamente bons (*known good*) como os do sistema operacional, diminuindo o escopo que o perito deve trabalhar. O processo é bem simples, uma vez com o *hashset* dos arquivos do disco investigado, o perito precisa apenas compara-los com os *hashsets* de arquivos conhecidos.

Para evitar ou atrapalhar esse processo o rootkit teria que ter o mesmo *hash* de um arquivo dito como bom. Seguindo os preceitos do paradoxo do aniversário que mostra a probabilidade de duas pessoas numa sala terem o mesmo aniversário, Preshing (2011) apresenta a formula: $1 - e^{-\frac{k(k-1)}{2N}}$, onde N é a quantidade de valores distintos que o *hash* pode apresentar e k é o tamanho do valor sequencial comparado, exemplo na sequencia: 0001₂, 0010₂, 0011₂ e 0100₂, o tamanho seria 0100₂. A tabela 4.2 mostra probabilidade de uma colisão de *hash* numa repetição uniforme e sequencial.

³<http://www.nsrl.nist.gov/Downloads.htm>

32-bit	64-bit	160-bit	Probabilidades
77163	5,06 bilhões	$1,42 \times 10^{24}$	50%
30084	1,97 bilhão	$5,55 \times 10^{23}$	1 em 10
9292	609 milhões	$1,71 \times 10^{23}$	1 em 100 ou um full house no poker
10	607401	$1,71 \times 10^{20}$	1 em 100 milhões ou ser mordido por um tubarão

Tabela 4.2: Probabilidade de uma colisão de hash em números sequenciais (PRESHING, 2011).

O *hash* de um valor arbitrário (como o rootkit) coincidir com o *hash* de um arquivo bom é improvável, contudo não impossível, como pode ser visto no trabalho dos chineses Xiaoyun Wang e Hongbo Yu da Universidade de Shandong em 2005 (WANG; YU, 2005), no trabalho eles desenvolvem um algoritmo que pode ser usado para criar arquivos arbitrários com o mesmo *hash* MD5, mas isso pode ser facilmente resolvido utilizando mais de um algoritmo como por exemplo SHA1 e MD5.

Outra forma que o rootkit pode se valer para anular essa técnica, é alterar partes não significantes de arquivos do computador, como na mensagem mostrada do cabeçalho da *PE*, pois mudar qualquer uma das letras da mensagem altera também o *hash* do mesmo, contudo isso fugiria de uma regra básica dos rootkits que é ficar indetectável. No fim das contas esse método é recomendado por sua grande utilidade e dificilmente um rootkit tentará ou conseguirá pervertê-lo, entretanto vale as ressalvas apresentadas.

4.6 Análise estática dos executáveis suspeitos desconhecidos

Análise estática é a análise do código de determinado malware para se ter uma melhor compreensão sobre suas funções, características e objetivos (DISTLER, 2007). Depois de filtrar os arquivos conhecidos o perito deverá iniciar a análise estática de alguns arquivos suspeitos. Certamente analisar todos os arquivos desconhecidos levaria muito tempo, inviabilizando na maioria dos casos a perícia.

O ideal nesse momento é filtrar dentre os arquivos desconhecidos os que são potencialmente maliciosos, para isso o especialista deverá ficar atendo principalmente a binários. Arquivos suspeitos são encontrados em todas as fases apresentadas, principalmente na recuperação de arquivos e na coleta de metadados, essas fases podem ajudar e muito a levantas suspeitas de arquivos, por exemplo, porque um arquivo de imagem com assinatura de jpg tem 2GB?

A fase onde são retirados os arquivos conhecidos, pode revelar programas conhecidamente maliciosos, esses arquivos podem ser investigados nessa fase também, para responder a per-

gunta primordial que é: esse código malicioso poderia subverter o(s) sistema(s) analisado? A resposta vai depender muito do tipo de ambiente analisado, pois se o ambiente for muito seguro, dificilmente uma ameaça qualquer conseguiria persistir, além do fato de que as chances de um rootkit *APT* estar em qualquer base de arquivos conhecidos é mínima.

De qualquer forma apesar da resposta óbvia ser em sua grande maioria verdadeira, o perito não pode aceitá-la resoluto se apegando a qualquer evidência solta ou mal compreendida, porque existe a chance, mesmo que mínima de que o artefato malicioso encontrado tenha sido plantado. Esse texto percorre sobre rootkit e sempre tenderá ao pior dos cenários, contudo cautela e bom senso é vital.

4.6.1 Packer

A antifofoense de processos de análise estática, consiste em impedir que os códigos do rootkit sejam analisados, para isso é comum que se use compreensão e criptografia nos arquivos do rootkit. Essa compreensão e/ou criptografia utilizada em qualquer código, sendo ele malicioso ou não, mexe com a estrutura do arquivo e o torna irreconhecível, de modo que não pode ser executado normalmente. Técnicas como essa são muito utilizadas por *malwares* para não serem detectados por antivírus, contudo sua aplicação também implica que sem uma estrutura padrão ele não pode ser interpretado pelo ambiente em que será executado e como já foi mostrado o rootkit em algum momento deverá ter uma estrutura padrão para que o mesmo possa ser executado. Para resolver esse problema existe o *stub* que será explicado abaixo.

Um *packer* é um programa que comprime um executável dentro de uma estrutura anormal e menor, de forma análoga a um programa de compactação de arquivos para o vulgarmente chamado *zip* só que com executáveis. Com Petri (2012) e Blunden (2009a) é possível entender como funciona um *packer*, mas especificamente o *Ultimate Packer for eXecutables (UPX)*⁴ que é um *packer* que suporta diversos tipos de executáveis de diversos ambientes como mostra a tabela 4.3.

Ao lidar com executáveis Windows principalmente *PE*, será encontrado uma estrutura com seções como *.text*, *.data*, *.idata* e *.fill*, como mostra a figura 4.4.

Depois de compactado o *UPX*, deixa as seções do executável como o da figura 4.5. O *UPX* combina todas as seções *.text*, *.data*, *.idata* etc em uma única seções chamada *upx1*, onde tem uma *stub* que descompacta o binário original.

O que o *UPX* faz é colocar o binário numa forma comprimida que o deixa com uma es-

⁴<http://upx.sourceforge.net/>

Nome completo	Descrição
amd64-linux.elf	Linux ELF
arm-wince.pe	Windows CE executable or DLL
fat-darwin.macho	Mac OS X executable
i086-dos16.sys	DOS 16-bit .sys file
i386-dos32.tmt.adam	DOS 32-bit executable
i386-dos32.watcom.le	DOS 32-bit linear executable
i386-win32.pe	Windows 32-bit executable or DLL

Tabela 4.3: Amostra de executáveis suportados pelo UPX (UPX, 2013).

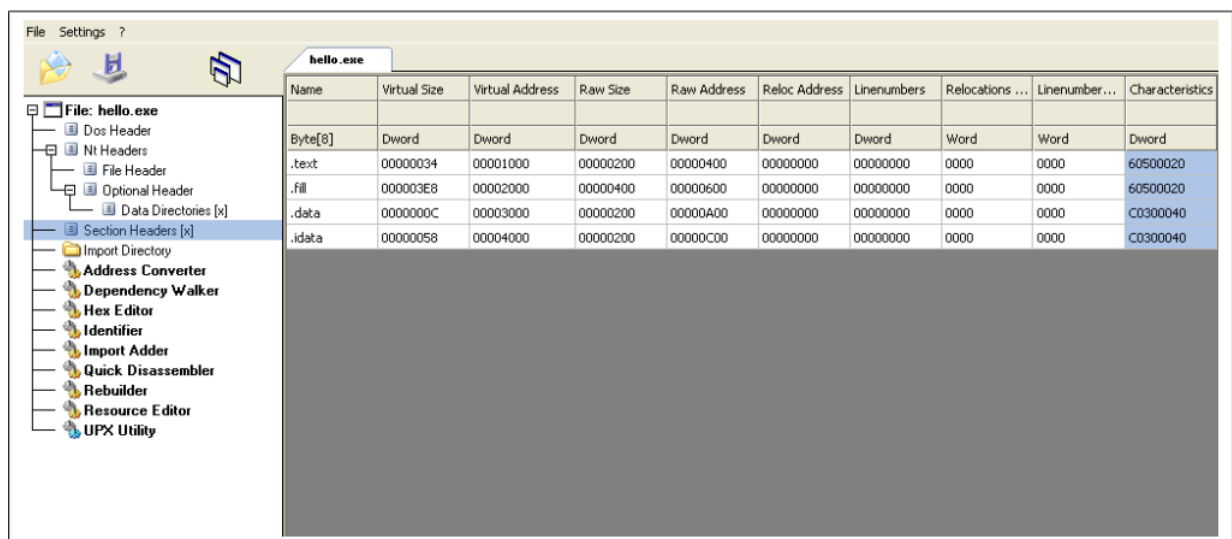


Figura 4.4: Hello World - PIMAGE SECTION HEADER (PETRI, 2012)

estrutura incomum ao ambiente de execução, logo as ferramentas de engenharia reversa como *disassemblers* não conseguem interpretar e consequentemente o sistema operacional também não, contudo o *UPX* também adiciona um executável capaz de descomprimir e executar o executável original em tempo de execução chamado de *stub*. O *stub* descompacta o *payload* que nesse caso é o executável original e muda o controle do executável para o *entry point* do binário original.

A seção *upx0* é basicamente espaço vazio que ocuparão na memória, o *upx1* é onde fica o *stub* e o executável original compactado. O *stub* irá descompactar o executável original o inserindo no espaço vazio utilizado pela seção *upx0* que ficam na *low memory* e executado-o, dessa forma o *stub* que fica logo acima na parte da memória também é sobrescrito durante o processo.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
UPX0	00005000	00001000	00000000	00000200	00000000	00000000	0000	0000	E0000080
UPX1	00001000	00006000	00000400	00000200	00000000	00000000	0000	0000	E0000040
UPX2	00001000	00007000	00000200	00000600	00000000	00000000	0000	0000	C0000040

Figura 4.5: Packed Hello World - PIMAGE SECTION HEADER (PETRI, 2012)

4.6.2 Bytecode

Outra forma de impossibilitar a análise estática, é compilar o rootkit em algum *bytecode*, que são interpretados por alguma *virtual machine* de forma análoga com código feito em Java e/ou .Net, ou seja, o código escrito em geral em uma linguagem de alto nível é compilado para algum tipo de *bytecode*, conhecidos como o *Java bytecode* ou o *.Net bytecode*, ou até mesmo algum *bytecode* privado. Em qualquer uma dessas possibilidades, o ambiente que o mesmo ira ser executado deverá ter a *virtual machine*, porque será ela que irá interpretar o *bytecode* e traduzir para instruções que o processador compreende.

As complicações para o uso dessa abordagem envolve que necessariamente o criador do rootkit terá que utilizar uma *virtual machine* para interpretar o *bytecode* em tempo de execução, caso seja feito em algum *bytecode* conhecido, o rootkit vai ser facilmente descompilado, mas se usar um *bytecode* privado o mesmo irá ter que criar a *virtual machine* privada também o que demanda muito tempo, mas em compensação dificulta muito o trabalho do perito, que antes de analisar o código terá que entender e descompilar a *virtual machine*. Da mesma forma que o código compactado, o código em *bytecode* não possui instruções conhecidas, logo ferramentas ou até mesmo o processador não o interpreta.

Esses foram dois métodos que podem ser utilizados para evitar ou complicar o processo de análise estática. Técnicas como essa podem ser usadas por programas legítimos também como é o caso do Skype (BIONDI; DESCLAUX, 2006). O importante é entender que não importa o que o rootkit pode usar, mas ele sempre vai ter que ter uma estrutura padrão para sua execução e que quanto mais baixo o nível de execução menos o rootkit pode fazer para se esconder, menos ele precisa efetivamente para se esconder e mais controle ele consegue do sobre o sistema

infectado, por exemplo, se um rootkit roda desde a inicialização da *BIOS*, quando o sistema operacional for executado o rootkit já está sendo executado.

4.7 Análise dinâmica dos executáveis suspeitos

As técnicas de antiforenses mostradas na análise estática podem ser combinadas dificultando ainda mais a análise e tornar o trabalho tão demorado que é proibitivo continuar, contudo o *stub* tem a chave para decodificar o binário, nesses casos a análise dinâmica pode acelerar o processo. O objetivo da análise dinâmica é o mesmo da análise estática apesar dos métodos diferentes (DISTLER, 2007).

Existe muitas formas e técnicas aplicáveis a análises dinâmicas, como análise do tráfego, análise da memória, análise dos processos, análise do registro do Windows etc, essa análise é bem diferente da que pode ser feita no ambiente infectado (análise *live*), mesmo que sejam usadas técnicas e ferramentas bem parecidas, porque a primeira, como já foi dito, o rootkit tem todas as vantagens e pode ter total controle sobre o ambiente ao contrário do segundo tipo que o perito prepara o ambiente e monitora. Nesse caso, quem chega primeiro tem todas as vantagens, seja o perito investigando um ambiente comprometido ou seja o rootkit executado em um ambiente preparado pelo perito.

Os *packers* também são um empecilho nessa fase e tudo fica mais complicado quando é usado um *packer* privado, pois nem todos se comportam como o *UPX*, de forma que o *packer* pode dividir o código em diversas seções e cada uma delas pode haver uma chave criptografada diferente ou então o rootkit pode ser programado para só executar no ambiente que ele infecta, por exemplo, ele pode usar verificações do hardware, ou ainda pode usar o *hardware* como parte da senha para descompactar e/ou decriptar o binário original.

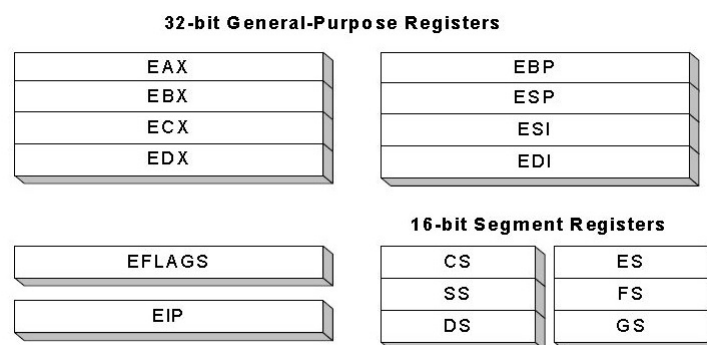


Figura 4.6: Registradores básicos IA32 (WISMAN, 2012)

O processador tem um registrador especial para *flags* na arquitetura Intel de microprocessa-

dores IA32 (figura 4.6), onde cada *bit* desse registrador representa um *flag* diferente. A diferença principal entre um processo comum e um que está sendo depurado é que este possui o *bit* 8 ou *Trap Flag (TF)* setada no registrador de *flags* chamado de *EFLAGS* (veja a figura 4.7).

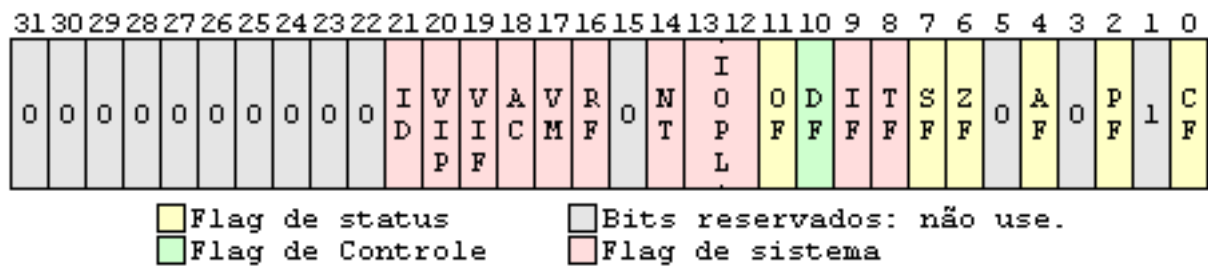


Figura 4.7: Registrador EFLAGS IA32 (WIKIDOT, 2010)

No modo de *debug* o processador irá executar uma instrução por vez, ignorando o paralelismo e *pipeline* forçando um ciclo de processamento completo antes de executar a próxima instrução, desse jeito o processador permite que os registradores, memória e instruções possam ser verificadas a cada passo (WIKIDOT, 2010).

O Windows tem métodos para verificar se o programa está sendo executado. O *Windows SDK* possui duas rotinas, *isDEBUggerPresent* em *user-mode* e *KdRefreshDebuggerNotPresent* em *kernel-mode* (FALLIERE, 2007), ou seja, o rootkit tem como identificar se está sendo depurado.

O uso de *virtual machines* facilitam muito o trabalho durante a análise. Contudo existe também formas do rootkit verificar se está num ambiente virtualizado e não executar. O rootkit pode ter todas essas técnicas para evitar ser analisado dinamicamente, contudo ele tem que executar de forma padrão. Ele pode e provavelmente irá verificar se está em um ambiente virtualizado ou se está depurado durante a execução, fazendo com que o perito precise mudar o fluxo de execução para forçá-lo a executar em tais circunstâncias.

5 *Conclusão*

O rootkit é um recurso sofisticado e valioso, pois seu desenvolvimento demanda tempo e profundo conhecimentos de arquitetura de computadores e de sistemas operacionais. Muitas das suas técnicas podem ser usadas por outros tipos de *malwares* e não necessariamente elas serão encontradas em todos os rootkits. Contudo dois aspectos são cruciais para definir um *malwares* como rootkit que são conseguir acesso root no sistema operacional e manter-se escondido.

Durante as etapas do processo foram apresentados algumas técnicas antiforenses, mas como não existe técnica antiforense que seja indetectável e nem metodologia para análise infalível, segura e totalmente confiável. Dessa forma as técnicas de antiforense assim como as de análise continuaram evoluindo e se confrontando numa eterna briga de gato e rato.

Foram mostrados diversas técnicas para atrasar e confundir o perito, e por vezes tornando o tempo de análise proibitiva. Em futuros trabalhos poderão ser abordados novas propostas e técnicas antiforense que poderão ser aplicadas a rootkits como o uso de *HPA* para persistir rootkits, esconder arquivos em dados em um *file system* próprio ou ainda um *packer* privado que decripta cada seção com uma chave diferente ligada a *hardware* do computador infectado. Também poderão ser desenvolvidas novas técnicas para análise forense de rootkits como automatizadores que agilizam o processo.

Esse estudo reuniu conceitos de rootkits e antiforense para desenvolver um processo geral para análise *post mortem* e seus empecilhos para ser usado em ambientes que podem ter sido subvertidos por rootkits.

Referências Bibliográficas

- BERINATO, S. *The Rise of Anti-Forensics*. 2007. Disponível em: <<http://www.csoonline.com/article/221208/the-rise-of-anti-forensics>>.
- BIONDI, P.; DESCLAUX, F. Silver needle in the skype. BlackHat Europe 2006, 2006. Disponível em: <<http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf>>.
- BLANCO, A.; EISSLER, M. One firmware to monitor é m all. Eko Party 2012, 2012. Disponível em: <<http://www.ekoparty.org/archive/2012/BlancoEissler.2012-paper.pdf>>.
- BLUNDEN, B. Anti-forensics: The rootkit connection. Black Hat USA 2009, 2009. Disponível em: <<http://www.blackhat.com/presentations/bh-usa-09/BLUNDEN/BHUSA09-Blunden-AntiForensics-PAPER.pdf>>.
- BLUNDEN, B. *The rootkit arsenal*. 1ª. ed. Plano, Texas: Wordware Publishing, 2009. ISBN 978-1-59822-061-2.
- BRASIL. Código de processo penal. 1941. Disponível em: <http://www.planalto.gov.br/ccivil_03/decreto-lei/del3689.htm>.
- BRASIL. Código de processo civil. 1973. Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/l5869.htm>.
- BUTLER, J. R.: The exponential growth of rootkit techniques. Black Hat, 2006. Disponível em: <<http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Butler.pdf>>.
- DISTLER, D. Malware analysis: An introduction. SANS Institute Reading Room, 2007. Disponível em: <http://www.sans.org/reading_room/whitepapers/malicious/malware-analysis-introduction_2103>.
- EASTTOM, C. Utilizing slack space for hidden partitions. 2007. Disponível em: <<http://chuckeasttom.com/ghostdrive.pdf>>.
- ECONOMOU, N. A.; JUAREZ, D. Vga persistent rootkit. Eko Party 2012, 2012. Disponível em: <http://www.ekoparty.org/archive/2012/VGA_Persistent_Rootkit.pdf>.
- FALLIERE, N. Windows anti-debug reference. Security Focus, 2007. Disponível em: <<http://www.symantec.com/connect/articles/windows-anti-debug-reference>>.
- FLORIO, E. When malware meets rootkits. Virus Bulletin, 2005. Disponível em: <<http://www.symantec.com/avcenter/reference/when.malware.meets.rootkits.pdf>>.
- FRANÇA, G. V. *Medicina Legal*. 8ª. ed. Rio de Janeiro - RJ: Guanabara-kooga, 2008. ISBN 978-85-7302-9635.

GUOJONSSON, K. Mastering the super timeline with log2timeline. SANS Institute Reading Room, 2010. Disponível em: <http://www.sans.org/reading_room/whitepapers/logging/mastering-super-timeline-log2timeline_33438>.

GUPTA, M. R. et al. Hidden disk areas: Hpa and dco. *International Journal of Digital Evidence*, 2006. Disponível em: <<http://www.utica.edu/academic/institutes/ecii/publications/articles/EFE36584-D13F-2962-67BEB146864A2671.pdf>>.

HALLMAN, M. *Timeline Creation and Analysis*. 2011. Disponível em: <<http://www.basistech.com/pdf/events/open-source-forensics-conference/osdf-2011-hallman-log2timeline.pdf>>.

HARRIS, R. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digital investigation 3S (2006) s4-s49*, 2006. Disponível em: <<http://www.dfrws.org/2006/proceedings/6-Harris.pdf>>.

HOGLUND, G.; BUTLER, J. *Rootkits: Subverting the Windows Kernel*. 1ª. ed. [S.l.]: Addison Wesley Professional, 2005. ISBN 0-321-29431-9.

HOUAISS, A.; VILLAR, M. de S. *Dicionário Houaiss da língua portuguesa*. 1ª. ed. Rio de Janeiro - RJ: Objetiva, 2009. ISBN 978-85-7302-9635.

INTEL. Intel 80386 programmer's reference manual 1986. Intel, 1986. Disponível em: <<http://css.csail.mit.edu/6.858/2011/readings/i386.pdf>>.

INTEL. Intel 64 and ia-32 architectures software developers manual, volume 3a: System programming guide, part 1. Intel, 2011. Disponível em: <<http://download.intel.com/design/processor/manuals/253668.pdf>>.

KASPERSKY. *Computer Security FAQ*. 2013. Disponível em: <http://www.kaspersky.com/threats_faq>.

LUDWIG, M. A. *The rootkit arsenal*. 1ª. ed. Show Low, Arizona: American Eagle Publications, Inc., 1995.

MACHADO, M. P.; ELEUTÉRIO, P. M. da S. *Desvendando a Computação Forense*. 1ª. ed. [S.l.]: Novatec, 2011. ISBN 978-85-7522-260-7.

MAYNOR, D.; MOOKHEY, K. K. *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. 1ª. ed. Rio de Janeiro - RJ: Syngress Publishing, Inc, 2007. ISBN 978-1-59749-074-0.

MCAFEE. McAfee threats report: Second quarter 2012. McAfee Labs, 2012. Disponível em: <<http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2012.pdf>>.

MCAFEE. *Glossary*. 2013. Disponível em: <<http://home.mcafee.com/virusinfo/glossary>>.

MCDUGAL, M. Live forensics on a windows system: Using windows forensic toolchest(wft). Fool Moon - Software Security, 2006. Disponível em: <http://www.foolmoon.net/downloads/Live_Forensics_Using_WFT.pdf>.

MEDINA, P. Österberg. Detecting rootkits in memory dumps. *PTS*, Associação Brasileira De Especialistas Em Alta Tecnologia (ABEAT), 2009. Disponível em: <<http://www.terena.org/activities/tf-csirt/meeting27/oesterberg-rootkits.pdf>>.

MESQUITA, F. I.; HOELS, B. W. P.; RALHA, C. G. Raciocínio baseado em casos aplicado em análise live. Associação Brasileira De Especialistas Em Alta Tecnologia (ABEAT), 2011. Disponível em: <<http://www.icofcs.org/2011/papers-published-007.html>>.

MICROSOFT. Welcome to the windows sdk. Microsoft, 2008. Disponível em: <[http://msdn.microsoft.com/en-us/library/ms717358\(v=vs.90\).aspx](http://msdn.microsoft.com/en-us/library/ms717358(v=vs.90).aspx)>.

MICROSOFT. Breakpoint syntax. Microsoft, 2013. Disponível em: <[http://msdn.microsoft.com/en-us/library/windows/hardware/ff538936\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff538936(v=vs.85).aspx)>.

MICROSOFT. *Microsoft PE and COFF Specification*. 2013. Disponível em: <<http://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx>>.

MULVEY, B. *Hash Functions*. 2007. Disponível em: <<http://home.comcast.net/~bretm/hash/>>.

NIST. Test results for digital data acquisition tool: Encase linen 6.01. National Institute of Justice, 2008. Disponível em: <<https://www.ncjrs.gov/pdffiles1/nij/224147.pdf>>.

NIST. Test results for digital data acquisition tool: Ftk imager 2.5.3.14. National Institute of Justice, 2008. Disponível em: <<https://www.ncjrs.gov/pdffiles1/nij/222982.pdf>>.

PETRI, D. Using upx as a security packer. 2012. Disponível em: <http://dl.packetstormsecurity.net/papers/general/Using_UPX_as_a_security_packer.pdf>.

PIETREK, M. *An In-Depth Look into the Win32 Portable Executable File Format*. 2002. Disponível em: <<http://msdn.microsoft.com/en-us/magazine/cc301805.aspx>>.

PRESHING, J. *Hash Collision Probabilities*. 2011. Disponível em: <<http://preshing.com/20110504/hash-collision-probabilities>>.

ROGERS, D. M. Anti-forensic presentation given to lockheed martin. San Diego, 2005.

ROSANES, P. Rootkits. *GRIS - Grupo de Resposta a Incidentes de Segurança*. Universidade Federal do Rio de Janeiro, 2011. Disponível em: <<http://www.gris.dcc.ufrj.br/documentos/artigos/rootkits-survey>>.

ROUSE, M. *DEFINITION: File format*. 2005. Disponível em: <<http://searchcio-midmarket.techtarget.com/definition/file-format>>.

RUBENS, P. *Buyer's Guide to Full Disk Encryption*. 2012. Disponível em: <<http://www.esecurityplanet.com/mobile-security/buyers-guide-to-full-disk-encryption.html>>.

RUSSINOVICH, M.; SOLOMON, D. A.; IONESCU, A. *Windows Internals 6^a Edition Part 1*. 6^a. ed. Redmond, Washington: Microsoft Press, 2012. ISBN 978-0-7356-4873-9.

RUTKOWSKA, J. Beyond the cpu: Defeating hardware based ram acquisition (part i: Amd case). Black Hat DC 2007, 2007. Disponível em: <<http://www.blackhat.com/presentations/bh-dc-07/Rutkowska/Presentation/bh-dc-07-Rutkowska-up.pdf>>.

SANTOS, C. M. *Dicionário Inglês-Português*. 2^a. ed. Santa Terezinha, São Paulo: Rideel, 2000.

THOMPSON, I.; MOROE, M. *Fragfs: An advanced data hiding technique*. Black Hat, 2006. Disponível em: <<http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Thompson/BH-Fed-06-Thompson-up.pdf>>.

TORRES, G. *Hardware Curso Completo*. 4^a. ed. Rio de Janeiro - RJ: Axcel Books do Brasil Editora, 2001. ISBN 85-7323-165-3.

UPX. *Supported executable formats*. 2013. Disponível em: <<http://upx.sourceforge.net/>>.

VELHO, J. A.; GEISER, G. C.; ESPINDURA, A. *Ciências Forenses*. 1^a. ed. Campinas - SP: Millennium, 2012. ISBN 978-85-7625-249-8.

WANG, X.; YU, H. *How to break md5 and other hash functions*. Shandong University, 2005. Disponível em: <<http://www.infosec.sdu.edu.cn/uploadfile/papers/How%20to%20Break%20MD5%20and%20Other%20H>>.

WIKI, F. *Timestomp*. 2007. Disponível em: <<http://www.forensicswiki.org/wiki/Timestomp>>.

WIKIDOT. *Registradores*. 2010. Disponível em: <<http://assemblytutorial.wikidot.com/registers>>.

WISMAN, R. *Syllabus C335 Computer Structures*. 2012. Disponível em: <<http://homepages.ius.edu/RWISMAN/C335/html/Chapter4.htm>>.