Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

ANÁLISES ESTÁTICA E DINÂMICA DE PROGRAMAS MALICIOSOS (MALWARES)

RAFAELA DONASSOLO ALVES

ORIENTADOR: PROF. DR. RUBENS BARBOSA FILHO

DOURADOS-MS
2020

ANÁLISES ESTÁTICA E DINÂMICA DE PROGRAMAS MALICIOSOS (MALWARES)

RAFAELA DONASSOLO ALVES

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Rafaela Donassolo Alves e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

ORIENTADOR: PROF. DR. RUBENS BARBOSA FILHO

DOURADOS-MS
2020

A482a Alves, Rafaela Donassolo
Análise estática e dinâmica de códigos maliciosos (malwares) / Rafaela Donassolo Alves. – Dourados, MS: UEMS, 2020.
74p.

Monografia (Graduação) – Ciência da Computação – Universidade Estadual de Mato Grosso do Sul, 2020.
Orientadora: Prof. Dr. Rubens Barbosa Filho.

1. Malware 2. Código malicioso 3. Análise estática I. Barbosa Filho, Rubens II. Título

CDD 23. ed. – 005.84

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

ANÁLISES ESTÁTICA E DINÂMICA DE PROGRAMAS MALICIOSOS (MALWARES)

RAFAELA DONASSOLO ALVES

Novembro de 2020

Banca Examinadora:
Prof. Dr. Rubens Barbosa Filho (Orientador)
Área de Computação – UEMS
Profª. Drª. Mercedes R. Gonzales Márquez
Área de Computação – UEMS
Prof. Dr. Osvaldo Vargas Jaques
Área de Computação – UEMS

AGRADECIMENTOS

Primeiramente, agradeço aos meus pais Aguinaldo e Vanessa, pelos exemplos de amor, companheirismo e dedicação que me oferecem a cada dia.

Agradeço a minha irmã Isabela que sempre está ao meu lado me apoiando em todas as minhas decisões.

Agradeço ao meu orientador, Prof. Dr. Rubens Barbosa Filho, pela competência e dedicação durante a elaboração desse trabalho.

Agradeço todos os professores do curso que me proporcionaram conhecimento, especialmente Prof. Dr. Odival Faccenda e Profª. Drª. Glaucia Gabriel Sass por me proporcionar minha primeira iniciação científica.

Agradeço aos meus amigos por todo carinho e apoio em tantos momentos difíceis desta caminhada.

RESUMO

Códigos maliciosos são programas de computadores que tem como objetivo criar uma vulnerabilidade no sistema e assim roubar dados e informações dos usuários, além de outros problemas que podem causar. Portanto o trabalho tem como objetivo estudar programas maliciosos, em que é realizado análises desses códigos através de análises estática e dinâmica, e assim conseguir classificá-los de acordo com o comportamento. As características são apresentadas, assim como as técnicas para análise dos códigos maliciosos e as ferramentas necessárias para o estudo dos mesmos.

Palavras-chave: malware, código malicioso, análise estática, análise dinâmica.

ABSTRACT

Malicious code are malicious programs that have the purpose to create a vulnerability in the system and then steal data and user information besides other problems that can be created. Therefore the aim of this project is to study malicious programs by performing the investigation of these codes, through static and dynamic analysis and then be able to classify them according to their behaviour. The characteristics are presented as well as the techniques for the analysis and the necessary tools for this study.

Key-words: malware, malicious code, static analysis, dynamic analysis.

Sumário

1 Introdução

Ano após ano verifica-se que o número de códigos maliciosos cresce substancialmente. No ano de 2019, de acordo com a Kaspersky (2020d), houve um aumento de 14% de objetos maliciosos. A maioria desses códigos maliciosos está voltado principalmente para o lucro financeiro. No entanto, na década de 1970, os ataques eram praticamente realizados com o objetivo de diversão.

Um terço dos ataques financeiros, no ano de 2019, teve o objetivo de atingir usuários corporativos. Essas ações visaram obter informações sobre contas bancárias ou afetar os recursos financeiros da empresa. O Brasil foi o quarto país mais atingido por esses tipos de ataques ficando atrás apenas da Rússia, Alemanha e China (KASPERSKY, 2020a).

Já em 2020, com a pandemia do coronavírus, os cibercriminosos aproveitaram o tema para realizar ataques e disseminar códigos maliciosos. Alguns desses ataques envolvia uma mensagem enviada por um aplicativo de mensagem disponibilizando uma suposta oferta de serviço *streaming* gratuito durante a pandemia do COVID-19. Também os cibercriminosos utilizaram campanhas com nome de uma farmácia, a qual envia uma fatura de compra de álcool em gel. Isso levou alguns usuários a clicarem no *link* malicioso (KASPERSKY, 2020c).

Além desses ataques, os cibercriminosos utilizaram *fake news* relacionadas ao auxílio emergencial para conseguir realizar esse tipo de ação. Segundo a Kaspersky (2020b), o Brasil se tornou o quinto país mais atingido por ciberataques durante a pandemia.

Portanto, tanto as ferramentas utilizadas quanto os códigos desenvolvidos estão mais profissionais. Este conjunto, ferramentas e códigos abrangem técnicas de ataque, infecção e propagação avançadas. Diante deste cenário, há a necessidade de se manter um constante aprimoramento das técnicas e ferramentas de detecção, identificação e neutralização desses códigos maliciosos. O caminho para alcançar esses intentos passa pelo entendimento dos processos de segurança e inovação das tecnologias necessárias para conhecer um código/programa que possa ser classificado como malicioso.

A evolução dos ataques por meio de *malwares* fez com que surgissem novos desafios para a área de segurança de computadores. Conhecer o funcionamento de um código malicioso é o ponto de partida para se produzir ou mesmo escolher as ferramentas adequadas para detecção e proteção dos usuários. Tais ferramentas adequadas permite conhecer o contexto em que o código malicioso pretende agir, sendo por meio de coleta de dados, danos ao sistema, fraudes financeiras ou ameaças e extorsões. O trabalho apresenta, no capítulo dois, um referencial teórico para o melhor entendimento do material de estudo. Depois é apresentado o conceito de máquina virtual no capítulo três. O capítulo quatro apresenta a metodologia científica junto com as ferramentas e os resultados dos estudo. Por fim, o capítulo cinco apresenta a conclusão desse trabalho.

1.1 Objetivo geral

Estudar arquivos maliciosos através das análises estática e dinâmica.

1.2 Objetivos específicos

- Estudo de técnicas de análises estática e dinâmica;
- Estudo das ferramentas usadas em cada tipo de análise;
- Análises estática e dinâmica de sete códigos maliciosos;
- Classificação dos sete códigos dentre sete categorias ou famílias de *malwares*.

2 Referencial Teórico

Em 2017, houve um ciberataque que atingiu vários países. O Brasil foi um dos países atingidos com 15 estados afetados. Esse ciberataque afetou hospitais, indústrias, empresas e serviços de telefonia. No Brasil, além dos sites do Ministério Público Federal e do Tribunal de Justiça de São Paulo, sistemas do Instituto Nacional do Seguro Social também foram atingidos (BRITTO; FREITAS, 2017).

Esses tipos de ataques ocorrem, pois programas maliciosos são espalhados a fim de atingir dispositivos conectados a rede. Segundo Mangialardo e Duarte (2015), programas maliciosos, que também são chamados de *malwares*, são programas criados com o intuito de realizar ações que prejudicam um computador. Os principais motivos para a criação desses programas são ganho financeiro e coleta de informações confidenciais.

Outra definição para esses programas é de Sihwail, Omar e Ariffin (2018). Segundo Sihwail, Omar e Ariffin (2018), *malware* é um *software* que é inserido em um sistema sem que o usuário saiba e que compromete as funções do computador, roubando dados e evitando controles de acesso. De acordo com Sihwail, Omar e Ariffin (2018), um *malware* pode ser:

- Vírus
  Esse tipo de *malware* se replica e insere seu código em outros programas. Ele consegue se espalhar de um programa para outro e de um computador para outro.

- *Spyware*
  Esse tipo de código malicioso espia as atividades do usuário atingido e consegue obter informações sobre páginas *web* visitadas e número de cartão de credito sem a vítma saber e envia essas informações para o atacante

- *Botnet*
  Esse código malicioso controla remotamente um grupo de dispositivos como PCs e *smart phones*

- *Trojan Horse*
  Enquanto atua como um programa legítimo, esse código executa funções desconhecidas e indesejadas.Também permite que atacantes tenham acesso a máquina e conseguem informações confidenciais.

Além dessas categorias de *malware* apresentadas, também existem outros tipos como *Adware*, o qual traz constantemente propagandas para o computador, e *Ransomware*, o qual permite que o *hacker* bloqueie as informações do computador, isto é, esse programa

malicioso encripta os dados mais importantes e pede um pagamento para liberá-los. Portanto, com a variedade de *malwares*, é importante compreender o comportamento de cada um para assim classificá-los.

De acordo com Paulista e Marchi (2017), é necessário realizar a análise do comportamento de programas maliciosos, uma vez que auxilia no desenvolvimento de sistemas mais seguros. Paulista e Marchi (2017) ainda afirmam:

> Milhares de códigos maliciosos são criados diariamente e para tanto, se faz necessário analisá-los. A análise do código e comportamento malicioso visa o entendimento profundo do funcionamento de um malware(Filho e outros, 2009), e a partir desta análise é possível projetar sistemas mais seguros e nos proteger de ataques futuros. Além disto, para realizar detecção de malware é necessário conhecer seus possíveis comportamentos e assim concluir se um dado programa é benigno ou malicioso.

Assim para realizar o estudo de programas maliciosos, é importante compreender as técnicas utilizadas para análise. Portanto, existem duas técnicas a fim de obter as características e comportamentos dos *malwares*: análise estática e análise dinâmica. Segundo Leite (2016), uma análise estática consiste no estudo de um arquivo do tipo *Portable Executable*-(*PE Files*), isto é, arquivos executáveis, sem precisar executá-los. Nesse processo de análise, é possível identificar padrões nas *strings* utilizadas, sequência de *bytes*, distribuição de *opcodes*, entre outros.

Já a análise dinâmica é realizada ao executar o *malware*. Segundo Prado, Penteado e Grégio (2016), o código malicioso é executado em um emulador, ou seja, um ambiente controlado, por um período de tempo permitindo que a infecção aconteça. Assim, o *malware* mostra seu comportamento e dessa forma pode-se obter as ações realizadas pelo mesmo.

2.1 Análise Estática Básica

A análise estática básica, segundo Mangialardo e Duarte (2015), estuda o código utilizando programas antivírus, e analisando as *strings*, funções e obtém informações dos cabeçalhos dos programas.

2.2 Análise Estática Avançada

A análise estática avançada, de acordo com Mangialardo e Duarte (2015), o código estudado é desmontado e uma engenharia reversa é feita a fim de entender as ações do *malware* analisado.

2.3 Análise Dinâmica Básica

A análise dinâmica básica estuda o código malicioso em um ambiente controlado, em que os processos do *malware* serão monitorados e os pacotes de dados feitos pelo código serão analisados (PRAYUDI; RIADI; YUSIRWAN, 2015).

2.4 Análise Dinâmica Avançada

Essa análise, segundo Prayudi, Riadi e Yusirwan (2015), é um método aprofundado da análise dinâmica realizando *debugging* no arquivo, analisando o registrador e realizando uma análise do sistema *windows*.

2.5 Trabalhos Relacionados

Mangialardo e Duarte (2015) apresentaram um técnica para classificação de *malware* baseada na unificação das técnicas de análise estática e dinâmica com o aprendizado de máquina. Os autores iniciaram a classificação determinando os artefatos como malignos, para aqueles que são *malwares*, e benignos para aqueles que são não *malwares*.

Para realizar a classificação da análise estática, atributos como data de compilação, tamanho do arquivo, entropia, entre outros foram considerados. Na análise dinâmica, as APIs do sistema foram utilizados para realizar a classificação. Já na análise conjunta, a qual utiliza aprendizado de máquina, foram usados os algoritmos *C5.0* e *Random Forest*.

Mangialardo e Duarte (2015) realizaram nove experimentos, os quais têm o objetivo de definir o desempenho da identificação do *malware* e a classificação para os tipos de códigos maliciosos. A validação dos experimentos foi feita a partir da validação cruzada e as métricas consideradas foram acurácia, precisão, revocação e a medida-F.

Mangialardo e Duarte (2015) chegaram a conclusão em que a técnica unificada apresenta melhor acurácia que as técnicas isoladas. No método de classificação, em que foi utilizado o aprendizado de máquina, o melhor resultado foi com o algoritmo *Random Forest*. Além dessa técnica, os autores utilizaram a medida-F para medir o desempenho da classificação, e foi observado que o desempenho é melhor quando as técnicas de análise estática e dinâmica são aplicadas juntas.

Prado, Penteado e Grégio (2016) desenvolveram uma heurística para detectar um comportamento malicioso. Para isso o estudo utilizou a técnica de análise dinâmica. O modelo desenvolvido agrupa algumas característica e informações para definir uma ação suspeita. Assim, foram especificadas algumas características para uma heurística geral, e definido a detecção comportamental, em que *templates* seriam criados.

Prado, Penteado e Grégio (2016) utilizaram os *templates* criados para encontrar instâncias de comportamento de acordo com o algoritmo estabelecido. Para realizar os testes, os

autores pegaram amostras maliciosas e benignas. Assim, Prado, Penteado e Grégio (2016) obtiveram os resultados que mostraram que é viável aplicar heurísticas baseados nos comportamentos para detecção dos *malwares*.

Sihwail, Omar e Ariffin (2018) realizaram o estudo das técnicas para análise dos códigos maliciosos. Os métodos de detecção baseado em assinatura e detecção baseada em heurística foram apresentados. Sihwail, Omar e Ariffin (2018) citaram as técnicas de análise estática, dinâmica, híbrida e análise de memória.

Sihwail, Omar e Ariffin (2018) apresentaram a comparação de trabalhos que estudam exemplares de *malwares* e mostram as técnicas utilizados pelos arquivos maliciosos para evitar a detecção. Além de apresentarem as técnicas para evitar a detecção, Sihwail, Omar e Ariffin (2018) mostraram como que os códigos maliciosos conseguem acesso ao sistema alvo.

3 Máquina Virtual

Antes de qualquer análise envolvendo um código malicioso, é necessário configurar um ambiente seguro que impeça ao código malicioso infectar a máquina do analista. Uma máquina sem um configuração de segurança pode permitir que o código malicioso se espalhe facilmente para outras máquinas conectadas pela rede. Um ambiente seguro permite ao analista investigar o código malicioso sem expor a máquina de produção ou outras máquinas da rede a um risco desnecessário.

Atualmente, é possível utilizar uma máquina física dedicada ou uma máquina virtual no estudo de códigos maliciosos. Este trabalho utiliza uma máquina virtual em que é possível até mesmo executar o código malicioso. Uma Máquina Virtual possui uma estrutura onde se pode exemplificar como sendo um computador dentro de outro computador, conforme figura 1.

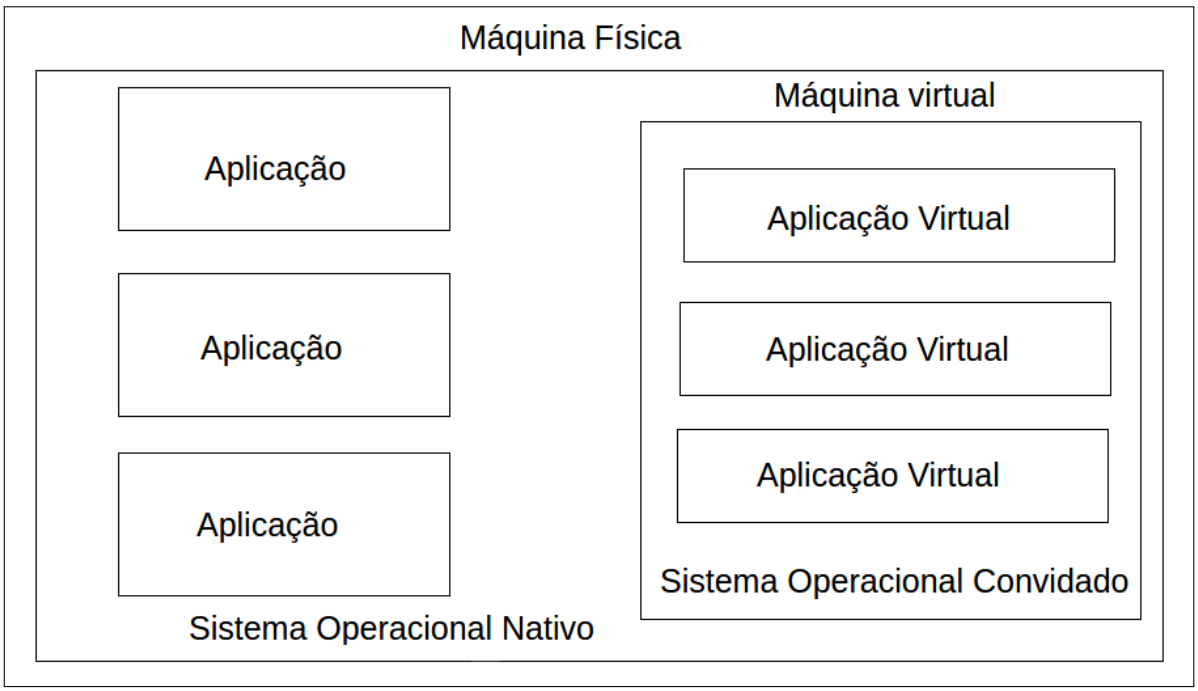


Figura 1 – Estrutura de uma máquina virtual

Próprio Autor

Ou seja, um computador possui um sistema operacional nativo e suas aplicações conforme representado pela figura 1. Quando uma máquina virtual é instalada, é possível que seja instalado um sistema operacional diferente do sistema nativo do computador. Assim, as aplicações e ações realizadas nesse sistema convidado não afetam o sistema operacional nativo da máquina.

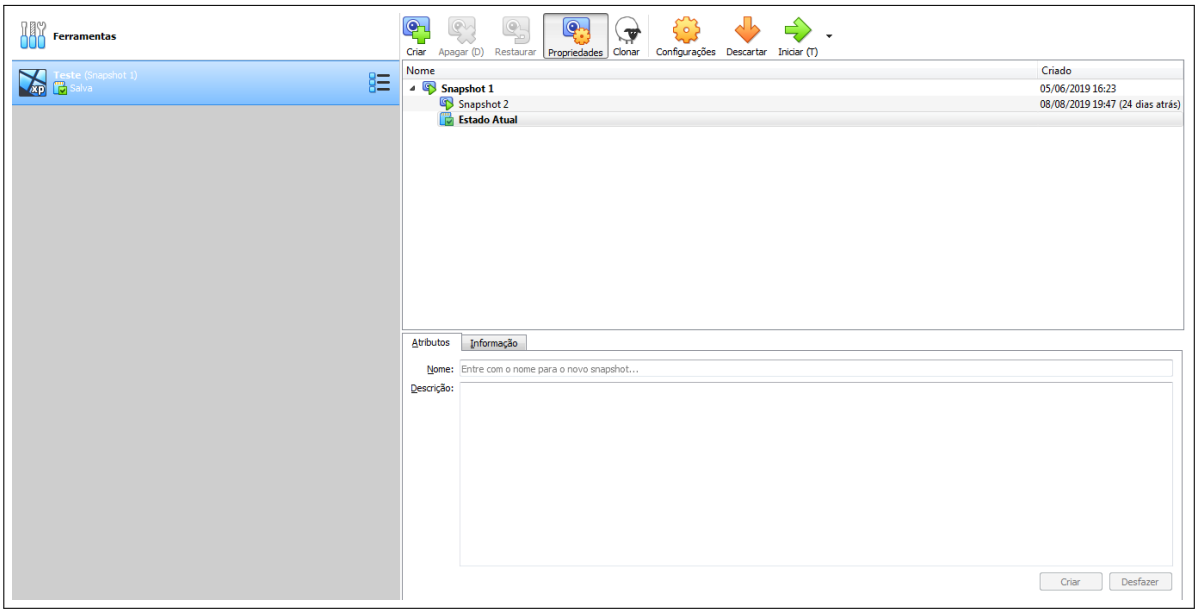


Figura 2 – Tela inicial da máquina virtual *VirtualBox*

Próprio Autor

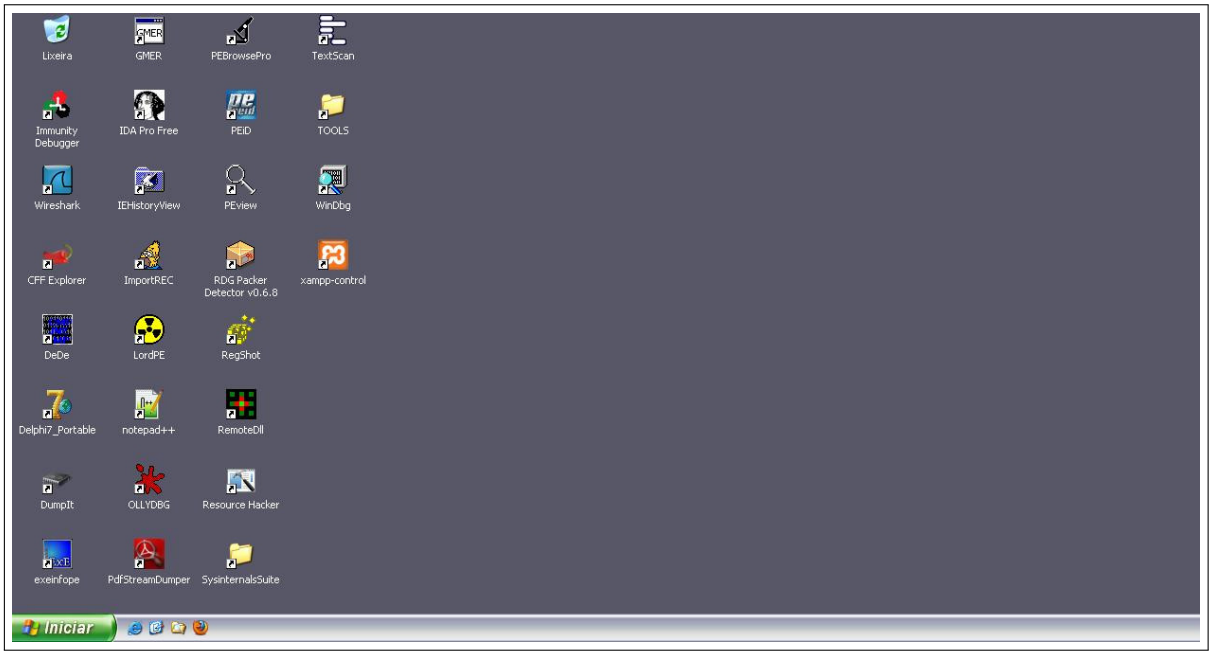


Figura 3 – Sistema Operacional *Windows XP* instalado na máquina virtual

Próprio Autor

A imagem 2 apresenta a máquina virtual *VirtualBox*. Na tela inicial é possível instalar mais de um sistema operacional. Já a imagem 3 apresenta um exemplo de sistema operacional instalado na máquina virtual *VirtualBox*, o qual é o *Windows XP*.

Também é possível que sejam feitos *snapshots* da máquina virtual, ou seja, é possível realizar "fotografia"do estado atual da máquina virtual. Assim caso ocorra algum problema, é possível restaurar a maquina virtual ao estado anterior. As imagens 4, 5 e 6 mostram o processo de restauração da máquina virtual.

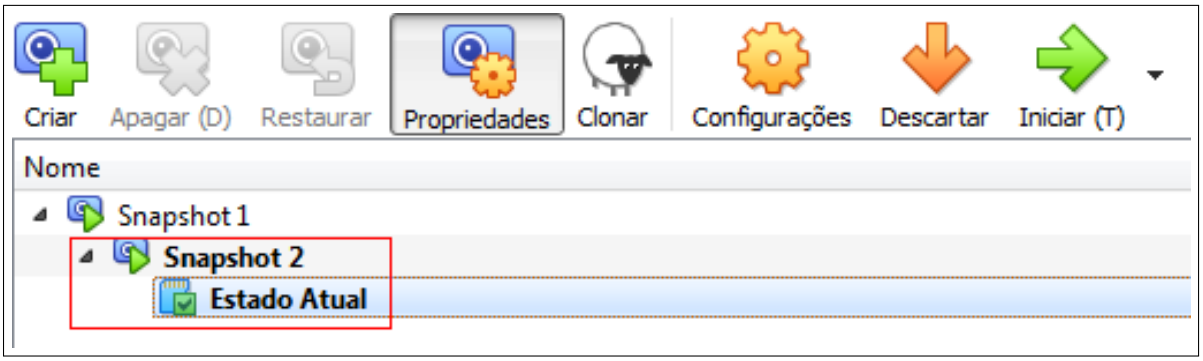


Figura 4 – Imagem que representa o estado atual da máquina virtual

Próprio Autor

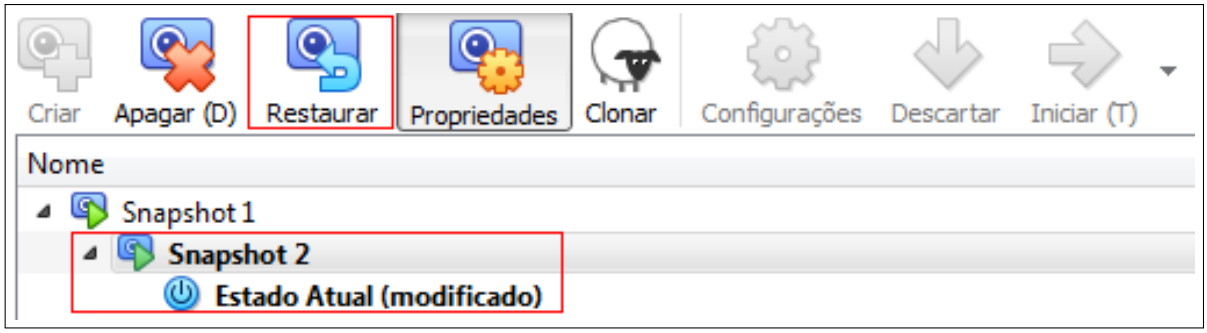


Figura 5 – Imagem que representa o estado modificado da máquina virtual

Próprio Autor

Figura 6 – Imagem que representa a restauração da máquina virtual

Próprio Autor

A imagem 5 apresenta o estado da máquina virtual modificado após a utilização da mesma. Assim é possível restaurar a máquina virtual ao estado anterior como a 6 apresenta.

4 Ferramentas

4.1 *BinText*

Essa ferramenta extrai *strings* ocultas em arquivos binários. *BinText* consegue extrair texto de qualquer tipo de arquivo e exibir texto *ASCII*, *unicode* e cadeias de recursos. A figura 7 abaixo mostra a tela da ferramenta com um arquivo aberto.

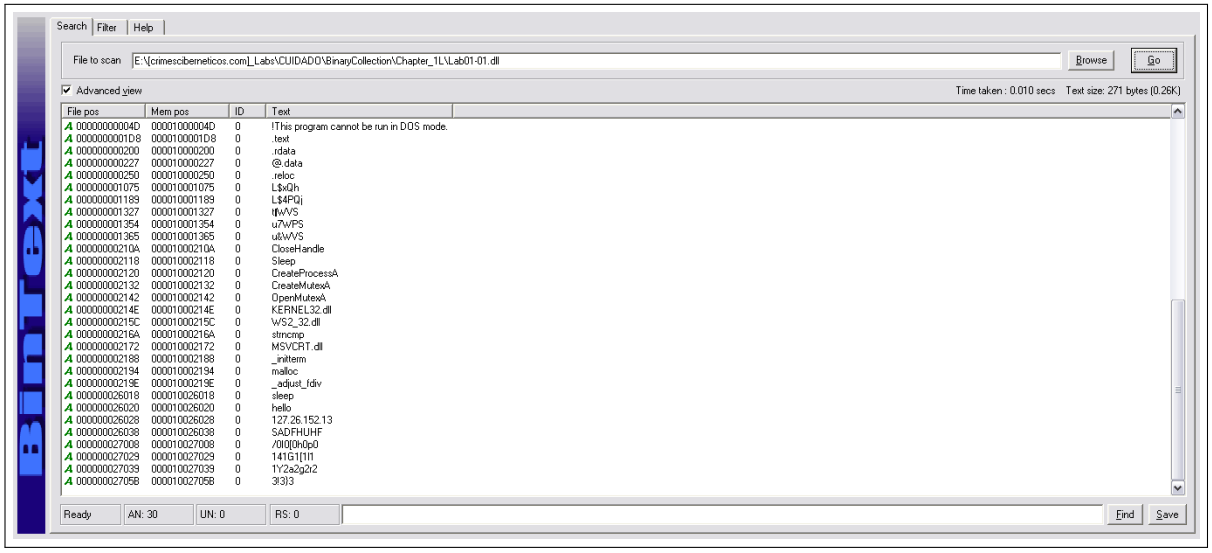


Figura 7 – Imagem do *BinText* com um arquivo aberto

Próprio Autor

4.2 *Delphi 7*

Delphi é uma linguagem de programação baseada na linguagem Pascal, e é utilizada para criar aplicações *mobile*, *desktop* e *Web*. A versão usada nesse trabalho é a 7. As imagens 55 e 56 na seção 5.6 apresentam a ferramenta.

4.3 *Delphi Decompiler*

Delphi Decompiler é uma ferramenta que possibilita realizar uma engenharia reversa no código, ou seja, quando não se tem o código original, é possível obter parte dele. A imagem 57 na seção 5.6 apresenta um exemplo do uso dessa ferramenta.

4.4 *ExeinfoPE*

Essa ferramenta possibilita analisar arquivos executáveis e ver todas as suas propriedades. Também é possível renomear arquivos, e obter informações como tamanho do arquivo e o ponto de entrada. A imagem 52 na seção 5.6 apresenta a ferramenta com um arquivo aberto.

4.5 *FakeNet*

FakeNet é uma ferramenta que simula uma rede falsa, ou seja, o *malware* executará suas funções sem infectar uma rede. Quando se utiliza o navegador, a ferramenta apresenta uma página padrão, isto é, pode-se alterar a *URL*, mas a página é a mesma. A imagem 8 mostra a execução da ferramenta no terminal e a figura 9 apresenta a página que a ferramenta cria.

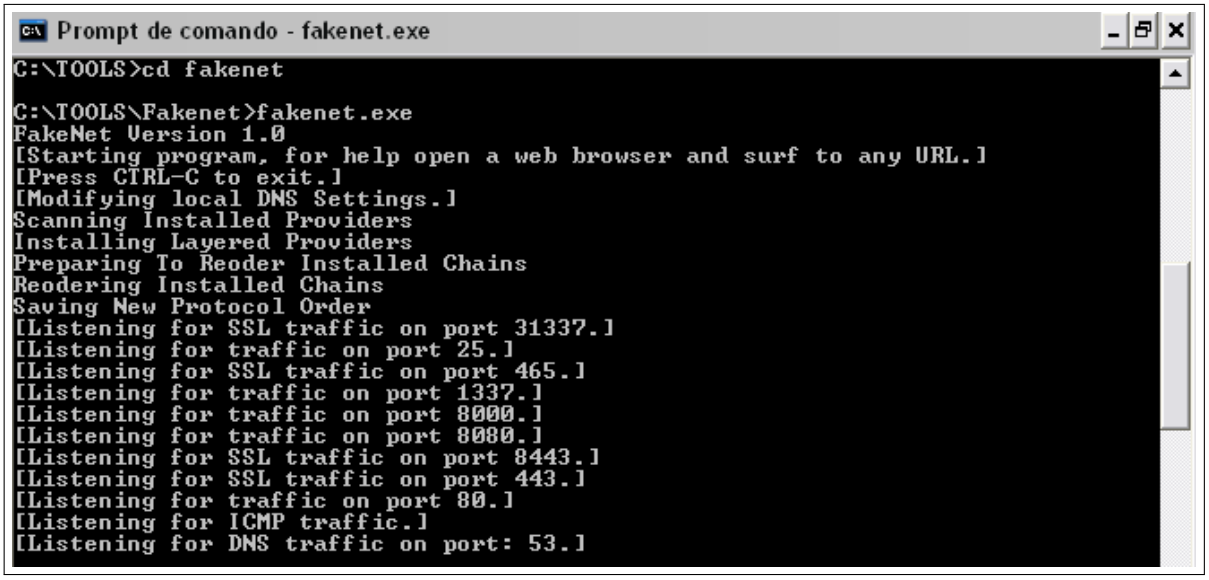


Figura 8 – Tela que apresenta a ferramenta inicializada no terminal

Próprio Autor

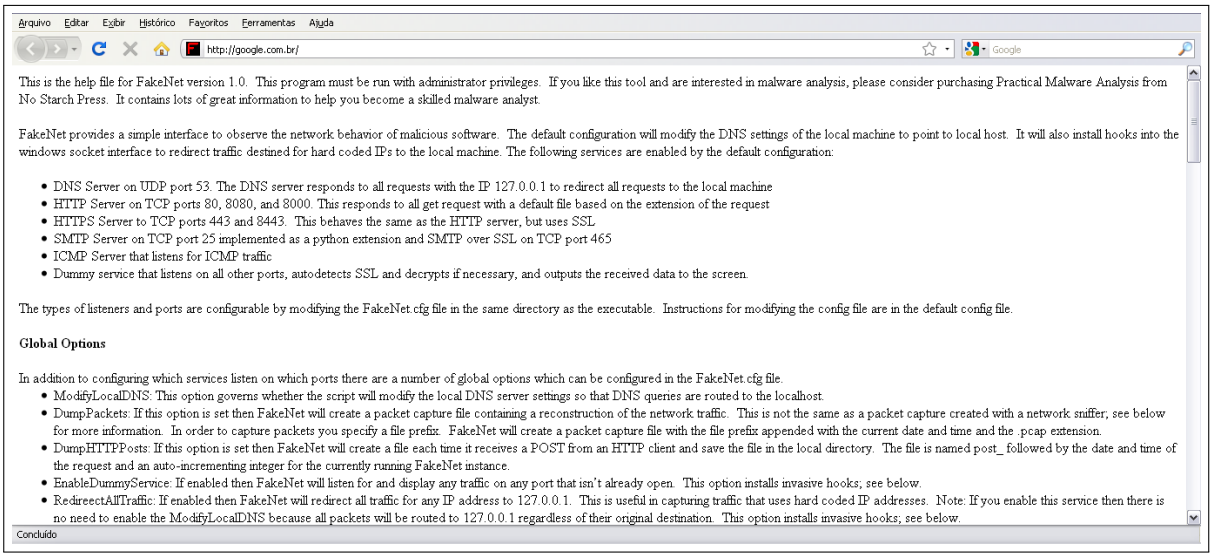


Figura 9 – Tela que apresenta o navegador com a página da ferramenta

Próprio Autor

4.6 *IDA PRO*

Essa ferramenta, segundo Sihwail, Omar e Ariffin (2018), exibe instruções *assembly*, fornece informações sobre o código analisado e extrai um padrão para identificar o programa malicioso. A imagem 10 apresenta a tela inicial da ferramenta.

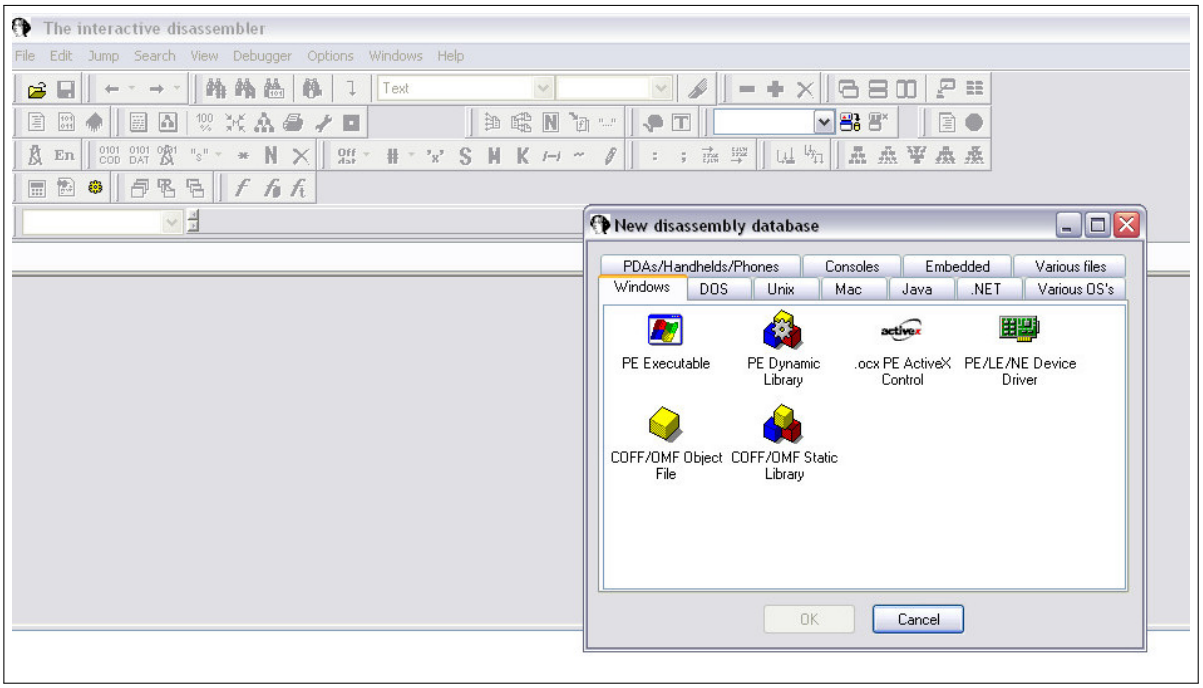


Figura 10 – Tela inicial da ferramenta *IDA Pro*

Próprio Autor

4.7 *OllyDBG*

Essa ferramenta é um *debugger* que é utilizada para monitorar a execução do comportamento de binários suspeitos em nível de instrução (SIHWAIL; OMAR; ARIFFIN, 2018). Segundo o Sihwail, Omar e Ariffin (2018), um *debugger* é um tipo de ambiente controlado, o qual é um programa que observa e examina a execução de outros programas binários. A figura 11 mostra um arquivo aberto na ferramenta.

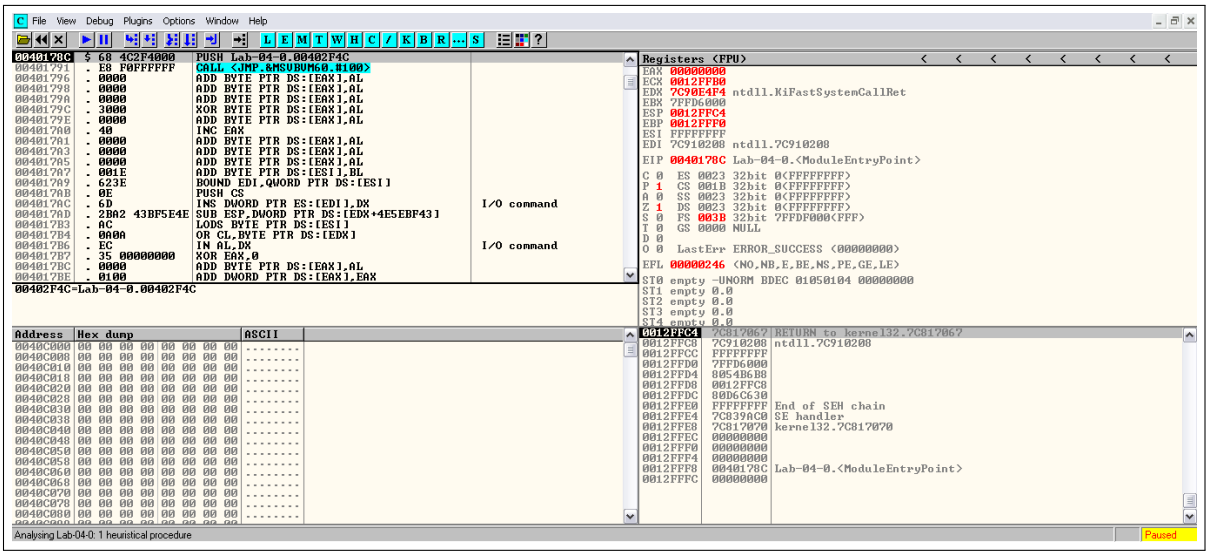


Figura 11 – Tela do *OllyDBG* com o arquivo analisado aberto

Próprio Autor

4.8 *PEiD*

Esse programa é utilizado para identificar o compilador utilizado pelo arquivo estudado e verificar se o código está empacotado. A figura 17 na seção 5.1 mostra a ferramenta com um arquivo aberto.

4.9 *PeView*

Nessa ferramenta é possível ver as informações do cabeçalho do arquivo e as diferentes seções do mesmo. Esse programa mostra informações sobre data de criação dos arquivos, as bibliotecas *DLLs* do mesmo e se o código acessa a rede mostrando endereço IP ou URL. A imagem 18 mostra o *PeView* com um arquivo aberto.

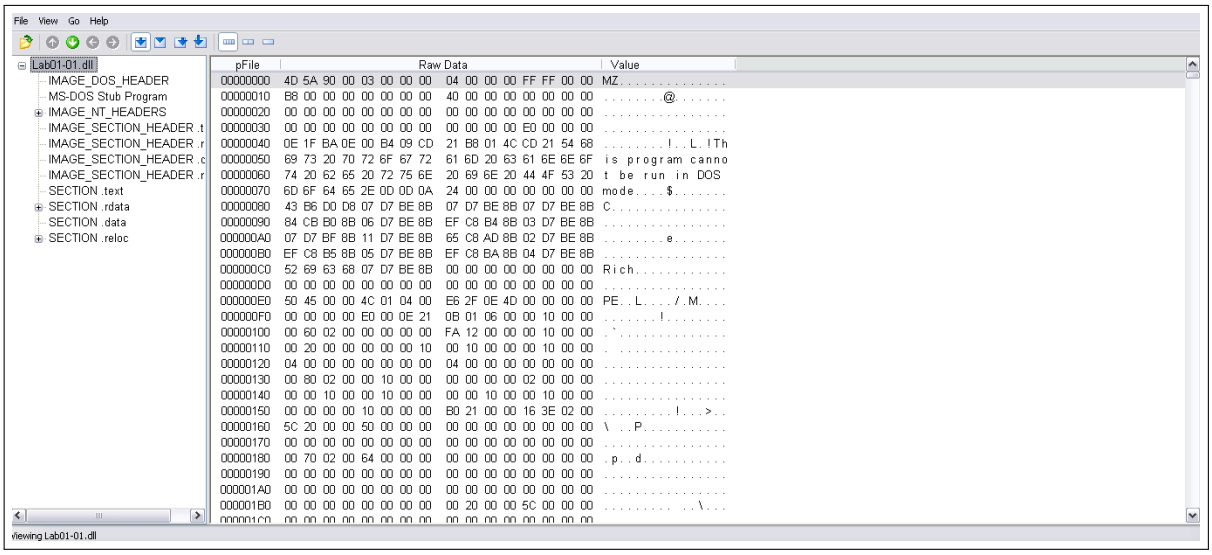


Figura 12 – Imagem do *PeView* com um arquivo aberto

Próprio Autor

4.10 *Process Explorer*

Essa ferramenta, de acordo com (MICROSOFT, 2020a), mostra informações sobre quais identificadores e processos de *DLLs* foram abertos ou carregados. A imagem 13 mostra o *Process Explorer*.

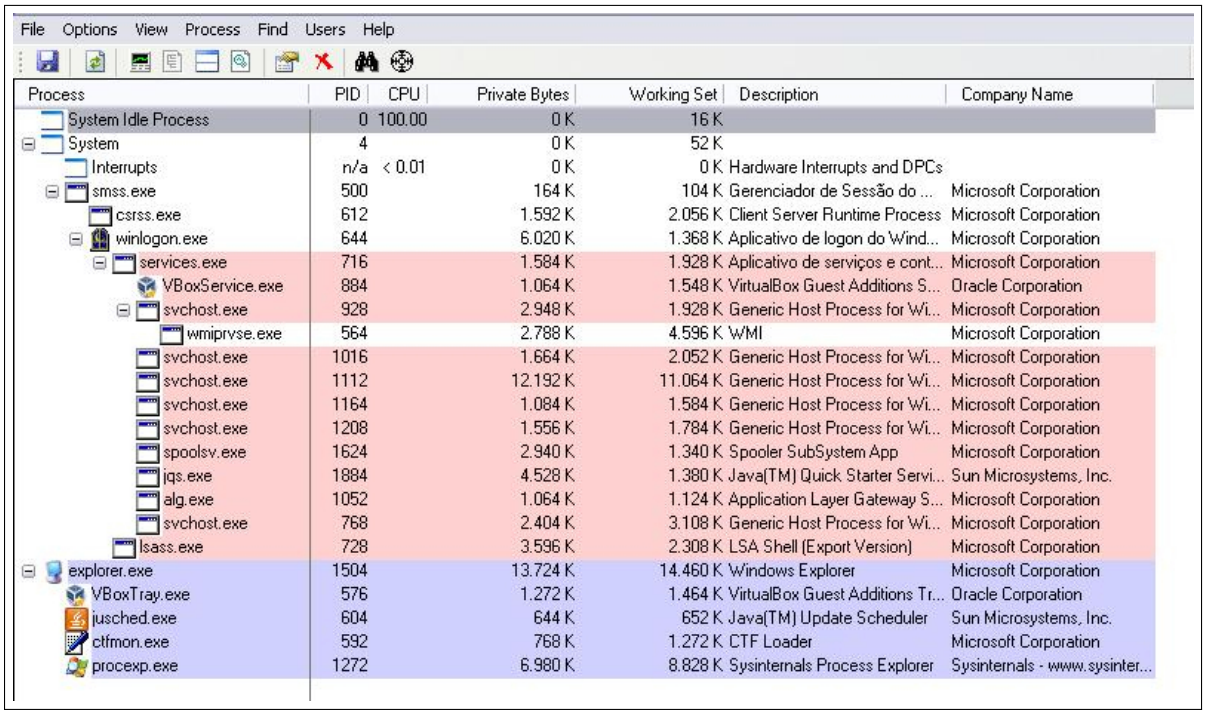


Figura 13 – Tela do *Process Explorer*

Próprio Autor

4.11 *Process Monitor*

Essa ferramenta é um programa de monitoramento, para *Windows*, a qual mostra em tempo real arquivos de sistema, registros, processos e *threads* (MICROSOFT, 2020b). A a imagem 14 mostra a tela inicial da ferramenta.

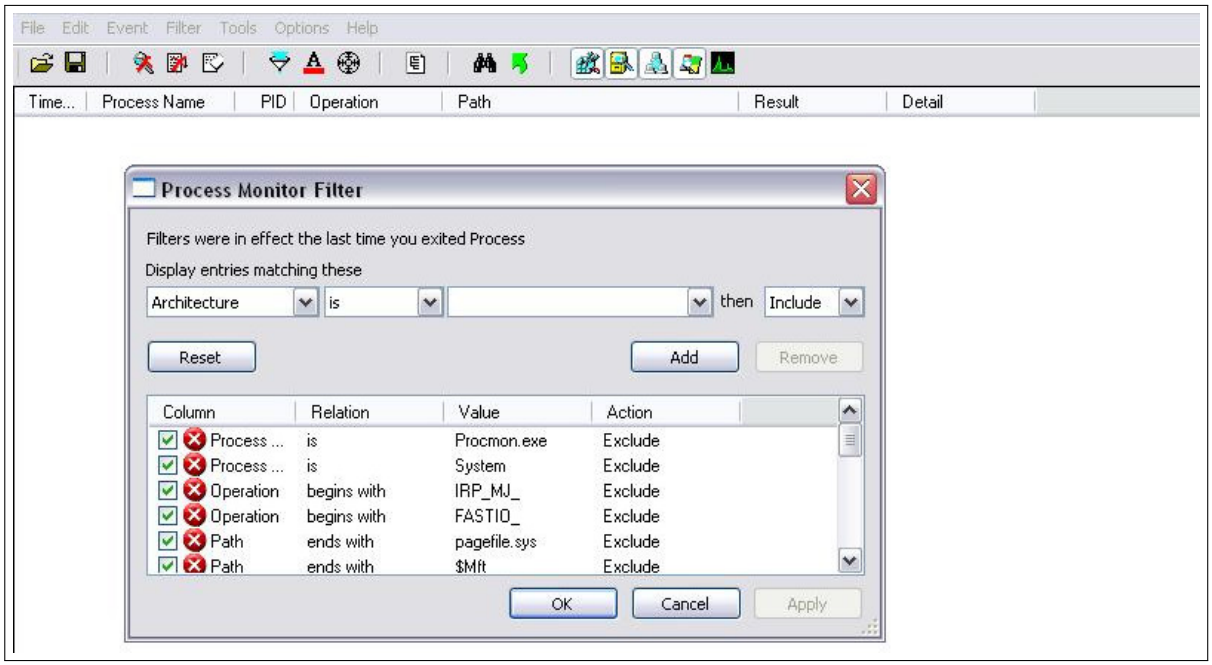


Figura 14 – Tela inicial do *ProcMon*

Próprio Autor

4.12 *Regshot*

O *RegShot* é utilizado para verificar mudanças no registro, ou seja, a ferramenta faz uma "fotografia"do registro antes e depois de executar o *malware* e assim mostrar as modificações feitas pelo código. A imagem 15 abaixo mostra esse programa.

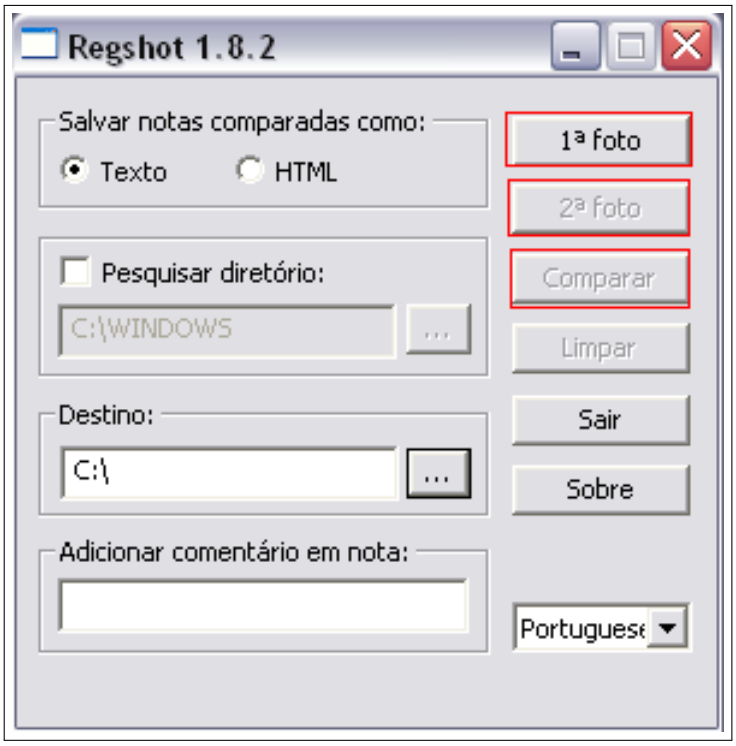


Figura 15 – Tela da ferramenta *RegShot*

Próprio Autor

4.13 *UPX*

Esse programa é utilizado tanto para desempacotar arquivos como empacotá-los. Nesse trabalho foi utilizada para desempacotar os arquivos analisados. O comando *upx* − *d* [*caminho do arquivo*] é utilizado para isso. A imagem 23 na seção 5.2 mostra a ferramenta desempacotando o arquivo analisado.

4.14 Virus Total

Esse *site* faz análise de arquivos passando por vários antivírus e mostrando quais desses programas conseguiram identificar o arquivo. Nessa ferramenta, é possível ver as propriedades básicas do arquivo, como o *hash* do programa, o tamanho do arquivo, o tipo do mesmo, e se utiliza empacotadores. Também tem outras informações que são possíveis ver, as quais são a data em que o arquivo foi criado, as bibliotecas e funções utilizadas pelo mesmo e se o arquivo faz conexão com a rede. A imagem 16 mostra a tela inicial do *site*.

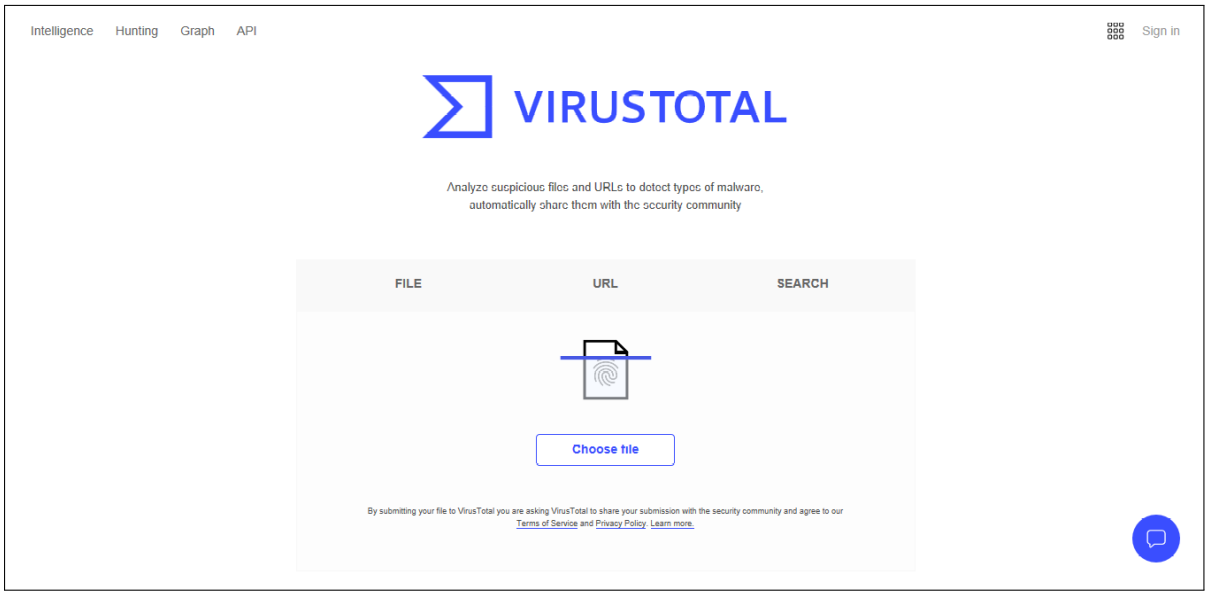


Figura 16 – Tela inicial do *site Virus Total*

Próprio Autor

5 Metodologia e Análises

O estudo requer um conhecimento científico, pois será realizado uma análise dos programas maliciosos para entender seu funcionamento e quais ações o código realiza na máquina atingida. Ao obter conhecimento do comportamento do programa malicioso, é possível identificar que tipo o mesmo se classifica. Portanto, para entender o comportamento do programa malicioso, os estudos serão realizados no laboratório do bloco de computação da Universidade Estadual de Mato Grosso do Sul (UEMS).

Neste trabalho é realizado um estudo bibliográfico sobre os tipos de *malwares* mais atuantes na comunidade de segurança. Estudou-se também as abordagens de análise estática e dinâmica de *malwares* assim como a utilização de ferramentas de apoio. Por fim são realizados sete laboratórios com características distintas que permitirão ao leitor identificar e compreender as técnicas de análise apresentadas nesse trabalho.

5.1 Arquivo disfarçado de biblioteca *DLL*

Para essa primeira análise foi utilizada a técnica de análise estática, a qual não é necessário executar o código estudado. Assim, esse primeiro estudo envolve dois arquivos, os quais um tem extensão *DLL* e o outro possui extensão *EXE*. Ambos passaram pelos programas *PeView*, *PEiD* e *BinText*. A primeira ferramenta utilizada foi o *PEiD* para verificar se o arquivo estava empacotado. De acordo com Sihwail, Omar e Ariffin (2018), *malwares* geralmente utilizam empacotadores para evitarem ser analisados. Assim foi visto que o código analisado não estava empacotado, como mostra a imagem 17.

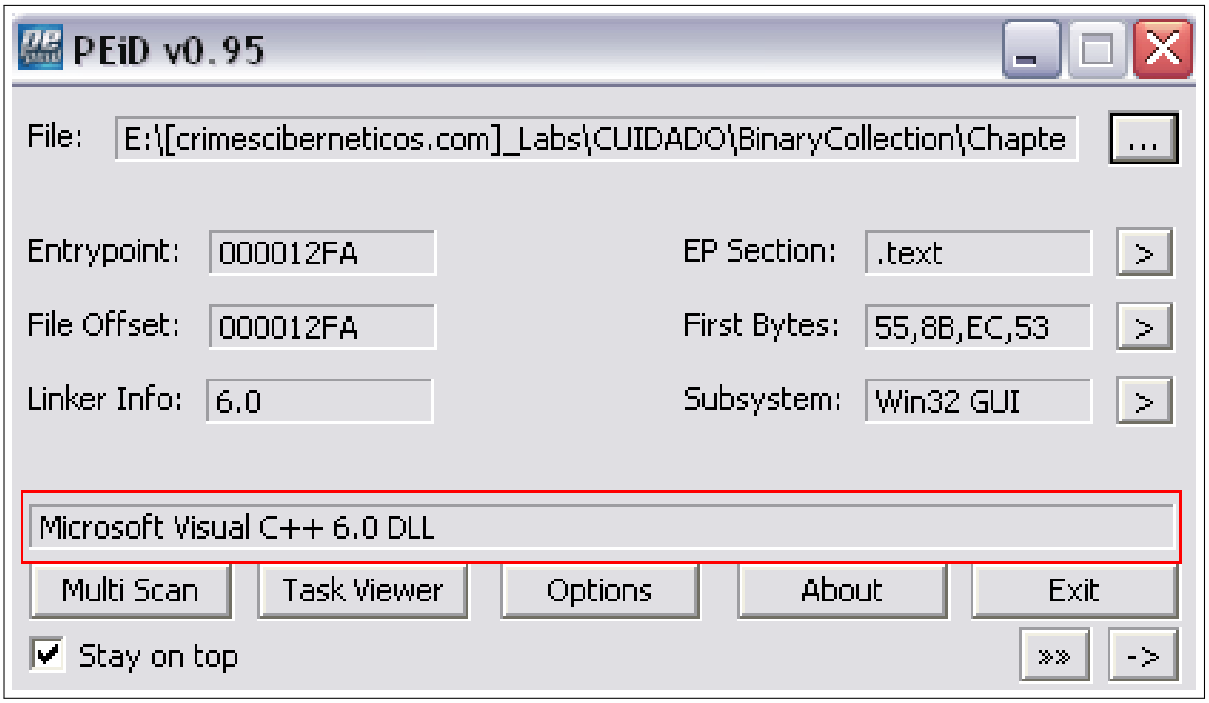


Figura 17 – Imagem do arquivo na ferramenta *PEiD* que mostra o compilador utilizado

Próprio Autor

Após esse processo, os dois códigos passaram pelo *PeView*. Nessa ferramenta, foi possível ver a data de criação dos dois arquivos analisados, ambos possuem a data de compilação em 19/12/2010, com pouca diferença no horário de criação. Como arquivos *DLLs* não executam sozinhos, acredita-se que os dois códigos fazem parte do mesmo pacote.

Foi possível identificar, também, as bibliotecas utilizadas pelos arquivos. Ambos fazem referência a *Kernel32.dll*, a qual lida com gerenciamento de memória, entrada e saída de operações e interrupções; e *Msvcrt.dll*, que é a biblioteca de tempo de execução C. O código *DLL* também utiliza a biblioteca *Ws2_32.dll*. Essa biblioteca fornece acesso a aplicações para funções Winsock e transporta fornecedores de serviços para carregar nomes e operações de mensagens (RUSSINOVICH; SOLOMON; IONESCU, 2012).

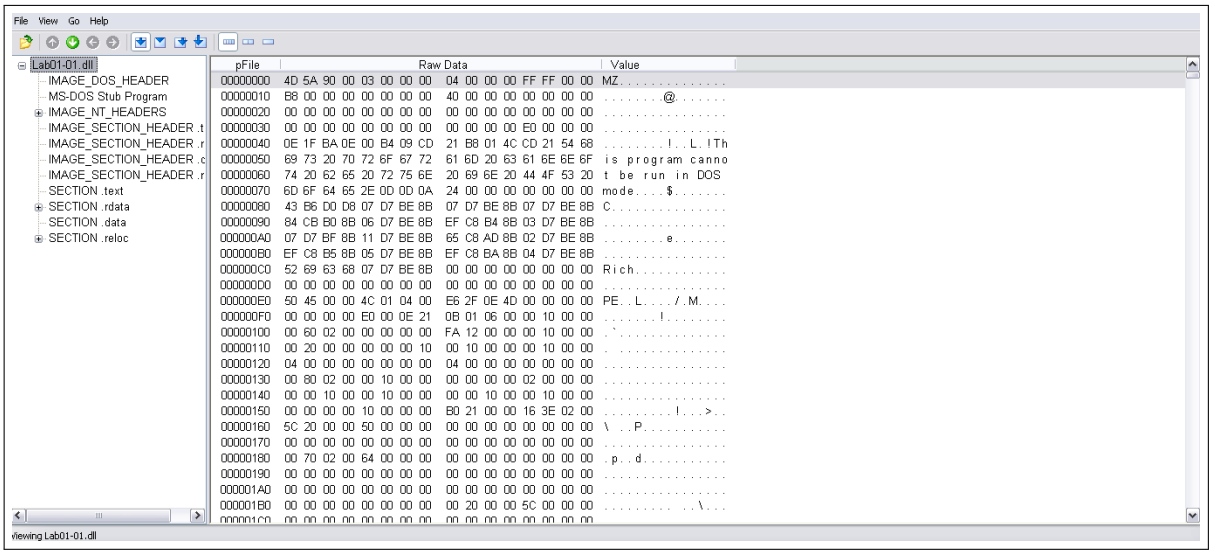


Figura 18 – Imagem do *PeView* com um arquivo aberto

Próprio Autor

Depois dessa análise inicial, foi utilizada a ferramenta *BinText*. Portanto, ao abrir os arquivos na ferramenta, foi possível identificar, no código executável, o endereço C:\Windows\System32\Kerne132.dll e C:\Windows\System32\Kernel32.dll, em que no primeiro possui alteração da letra l pelo número 1 para confundir o usuário. Já no arquivo *DLL*, foi visto que o mesmo faz referência a um endereço IP. A figura 19 mostra os endereços e a figura 20 indica o endereço IP.

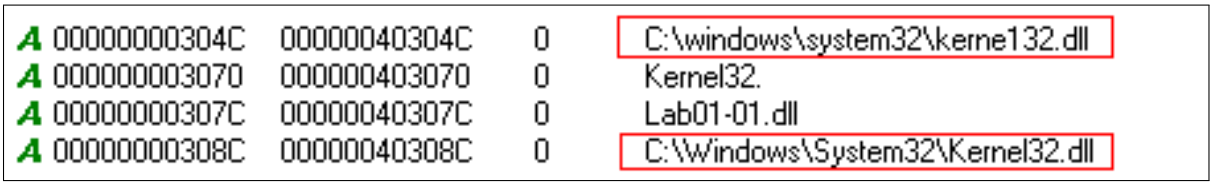


Figura 19 – Imagem do *BinText* que indica os endereços

Próprio Autor

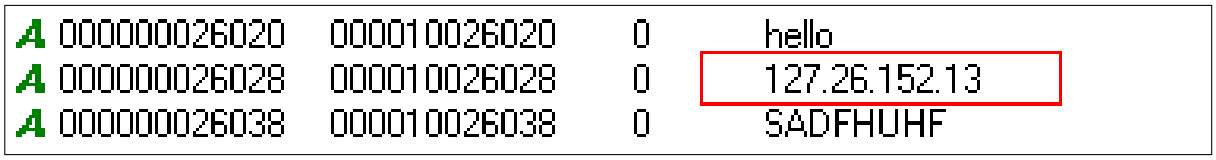


Figura 20 – Imagem do *BinText* que mostra o IP

Próprio Autor

Assim, o comportamento desse arquivo é parecido com o *malware Backdoor*, pois o programa tenta se esconder como se fosse um arquivo original do sistema e faz conexão com a rede. De acordo com Stallings e Brown (2015), um *Backdoor* consegue passar pela verificação de segurança de um sistema e permite acesso não autorizado a funcionalidades em um programa ou em um sistema comprometido.

5.2 Análise de arquivo executável empacotado

O segundo arquivo possui a extensão *EXE*, ou seja, é um executável. Para análise desse código foram utilizados o *site Virus Total*, as ferramentas *PeView*, *PEiD*, *UPX* e *BinText*. A técnica utilizada nesse estudo foi a análise estática. Portanto, para uma análise inicial, foi usado o *site Virus Total*.

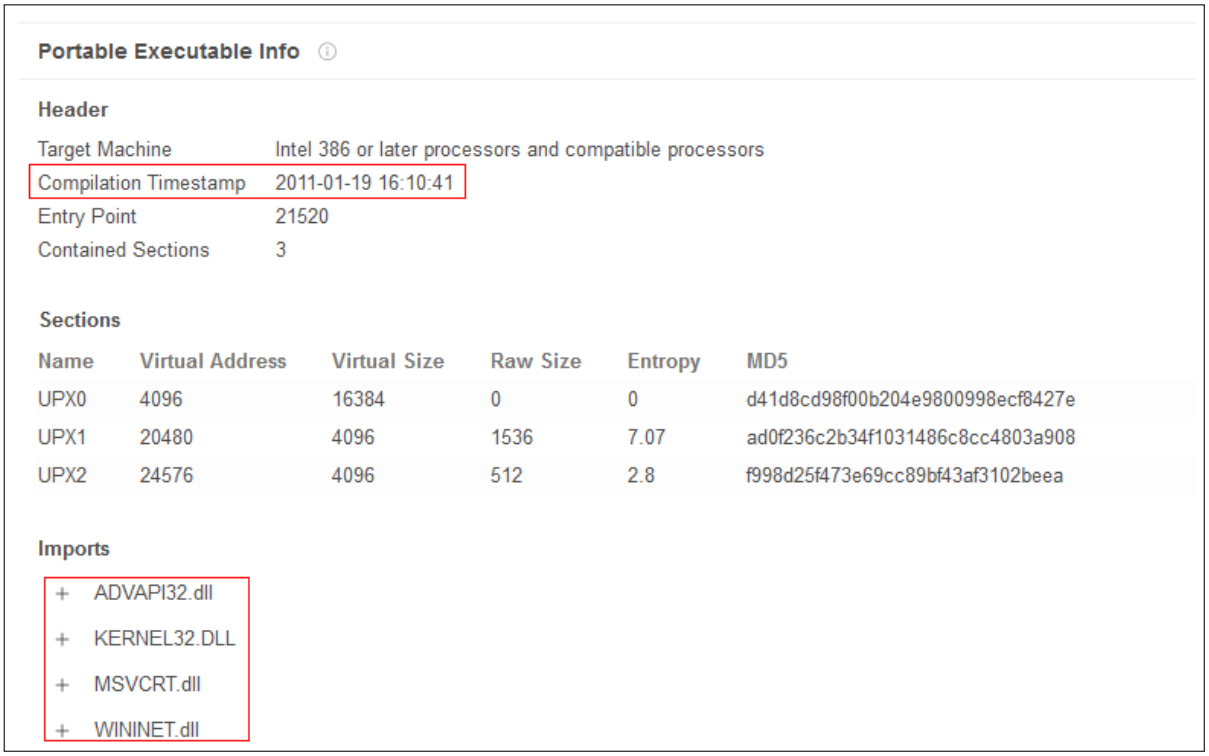
Portanto, para que se inicie a análise, é preciso que seja feito o *upload* do arquivo no *site*. Assim, foi visto que o código é empacotado, como mostra a imagem 21, e possui data de compilação em 19/01/2011. Também é possível obter informações sobre as bibliotecas utilizadas pelo arquivo, as quais são *Kernel32.dll, Advapi32.dll, Msvcrt.dll* e *Wininet.dll*. De acordo com Russinovich, Solomon e Ionescu (2012), a *Advapi32.dll* implementa toda a parte do cliente SCM APIs, e a biblioteca *Wininet.dll* permite que aplicações interajam com os protocolos *HTTP* e *FTP*. A imagem 22 mostra as informações sobre a data e as bibliotecas.

Basic Properties

MD5	8363436878404da0ae3e46991e355b83
SHA-1	5a016facbcb77e2009a01ea5c67b39af209c3fcb
SHA-256	c876a332d7dd8da331cb8eee7ab7bf32752834d4b2b54eaa362674a2a48f64a6
Vhash	03303e0f7d1019z601pz1bz
Authentihash	c0dd97382560a28cc053de86b9505ea78390147de7021744eb49d9b55e3d152f
Imphash	096aa05b8a2e1f2dc66fc73a1a978a7b
SSDEEP	48:atUKzxRhvlNZEVtfbn4m3ZUJSSeJY8JTalcLoBgs:0UKXktfb4KOJzcK
File type	Win32 EXE
Magic	PE32 executable for MS Windows (console) Intel 80386 32-bit
File size	3.00 KB (3072 bytes)
F-PROT	UPX
PEiD	UPX v0.89.6 - v1.02 / v1.05 -v1.24 -> Markus & Laszlo [overlay]

Figura 21 – Tela do *site Virus Total*, a qual indica que o arquivo é envelopado

Próprio Autor



Portable Executable Info

Header

Target Machine	Intel 386 or later processors and compatible processors
Compilation Timestamp	2011-01-19 16:10:41
Entry Point	21520
Contained Sections	3

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5
UPX0	4096	16384	0	0	d41d8cd98f00b204e9800998ecf8427e
UPX1	20480	4096	1536	7.07	ad0f236c2b34f1031486c8cc4803a908
UPX2	24576	4096	512	2.8	f998d25f473e69cc89bf43af3102beea

Imports

+ ADVAPI32.dll
+ KERNEL32.DLL
+ MSVCRT.dll
+ WININET.dll

Figura 22 – Tela do *site Virus Total*, que mostra a data de compilação e as bibliotecas

Próprio Autor

Depois de ver que o arquivo é empacotado, é necessário que o arquivo seja desempacotado para continuar a análise, uma vez que não é possível obter muitas informações com o mesmo envelopado. Assim, para realizar esse processo, foi utilizado o programa *UPX*, como mostra a imagem 23.

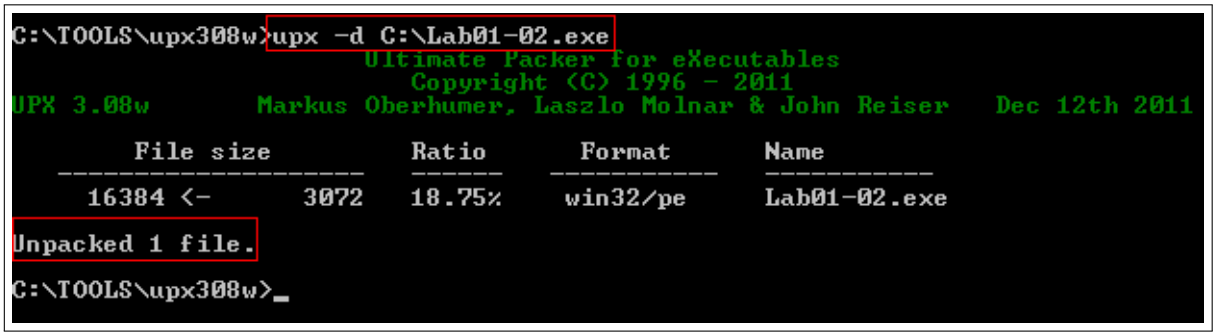


Figura 23 – Tela do arquivo submetido ao *UPX*

Próprio Autor

Após utilizar o *UPX*, o arquivo foi aberto pelo programa *PEiD*, e assim foi possível identificar a informação *Microsoft Visual C++ 6.0*, a qual indica o sobre o compilador utilizado pelo código. A imagem 24 mostra essa informação.

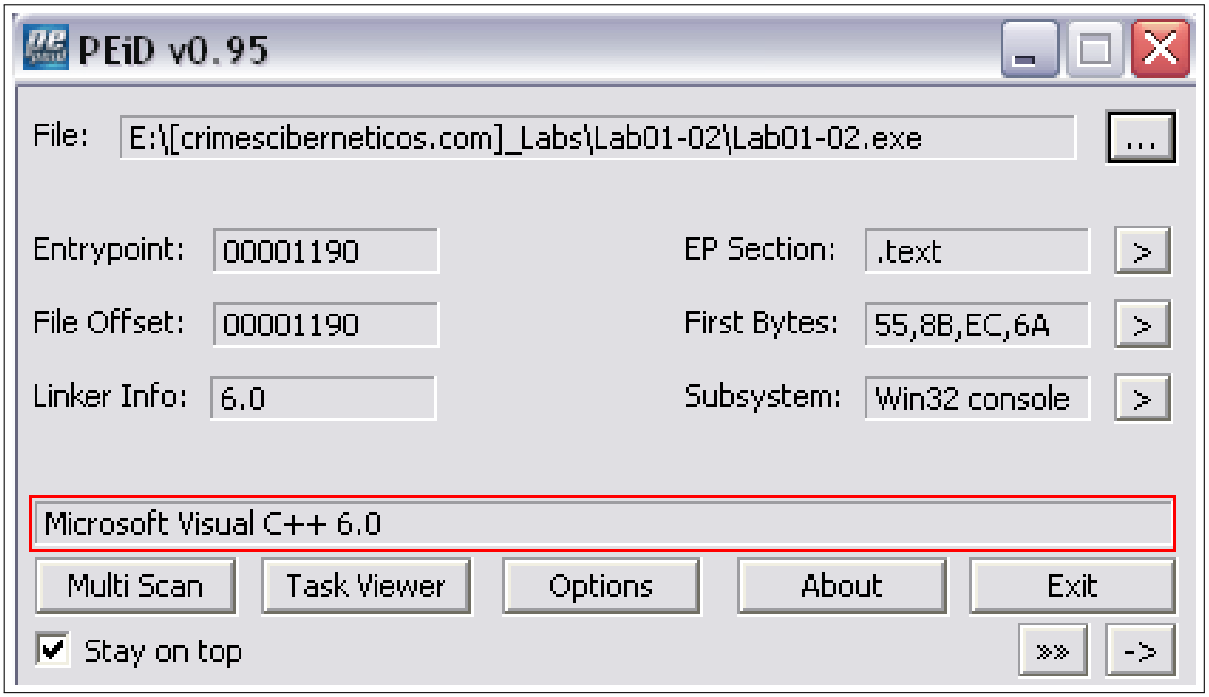
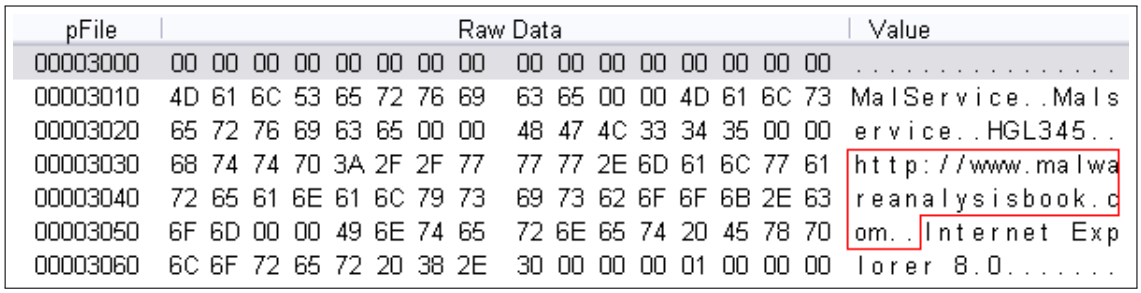


Figura 24 – Tela do arquivo após ser desempacotado e submetido ao *PEiD*

Próprio Autor

Depois de realizar esses procedimentos, o arquivo foi aberto pelo programa *PeView*. É importante destacar que o arquivo faz referência a uma *URL*, portanto o código faz uma conexão a rede. A imagem 25 mostra essa informação.



pFile	Raw Data	Value
00003000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00003010	4D 61 6C 53 65 72 76 69 63 65 00 00 4D 61 6C 73	MalService..Mals
00003020	65 72 76 69 63 65 00 00 48 47 4C 33 34 35 00 00	ervice..HGL345..
00003030	68 74 74 70 3A 2F 2F 77 77 77 2E 6D 61 6C 77 61	http://www.malwa
00003040	72 65 61 6E 61 6C 79 73 69 73 62 6F 6F 6B 2E 63	reanalysisbook.c
00003050	6F 6D 00 00 49 6E 74 65 72 6E 65 74 20 45 78 70	om..Internet Exp
00003060	6C 6F 72 65 72 20 38 2E 30 00 00 00 01 00 00 00	lorer 8.0.....

Figura 25 – Tela do *Peview* que mostra a *url* utilizada pelo código

Próprio Autor

Desse modo, ao analisar o comportamento do código, o mesmo pode ser classificado como *Drive-by-Download*, uma vez que o mesmo faz conexão com a rede e faz referência a uma *URL*. Segundo Stallings e Brown (2015), um *Drive-by-Download* utiliza seu código em um *website* comprometido para assim explorar uma vulnerabilidade no navegador para atacar um sistema de um cliente quando o *site* é visitado.

5.3 Análise de arquivo executável com *strings* criptografadas

Esse estudo envolve um programa, o qual tem extensão *SCR*, ou seja, é uma extensão de arquivos de *screen saver*. Para realizar essa análise foram utilizadas as técnicas de análise estática e dinâmica. O programa passou pelas ferramentas *PEiD*, *BinText*, *UPX* e *OllyDGB*. A primeira ferramenta utilizada nessa análise foi o *PEiD*. Assim, como mostra a imagem 26, a ferramenta indica qual programa para empacotar foi utilizado pelo código. Portanto para desempacotá-lo foi utilizado a ferramenta *UPX*, como a imagem 27 mostra.

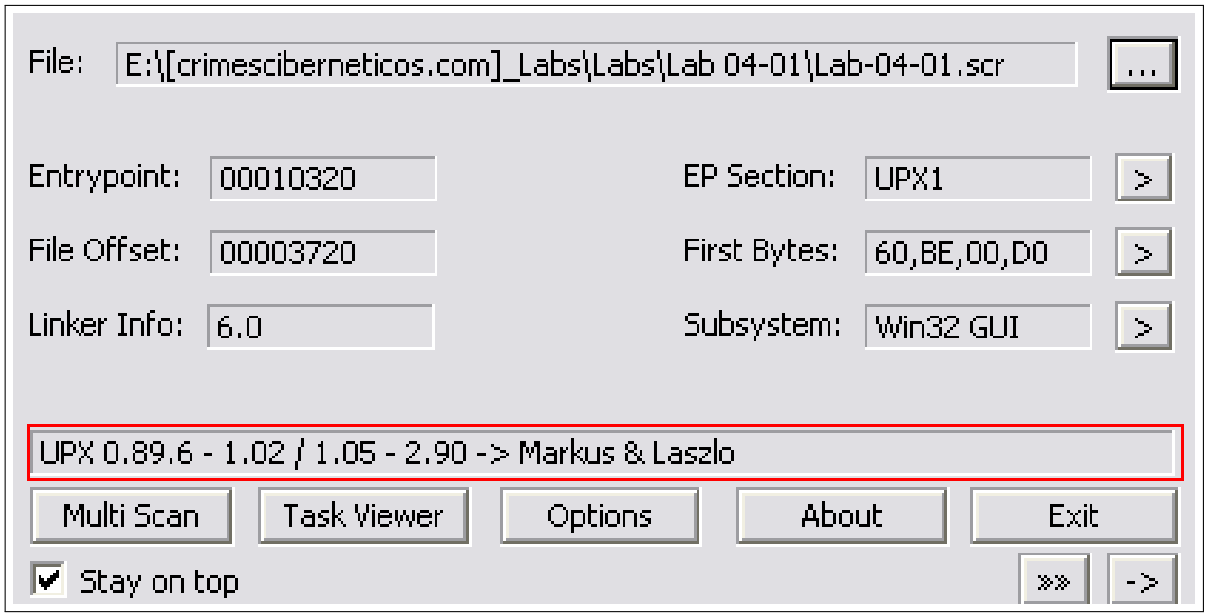


Figura 26 – Tela do *PEiD*, a qual indica que o arquivo é envelopado

Próprio Autor

Figura 27 – Tela do *UPX*

Próprio Autor

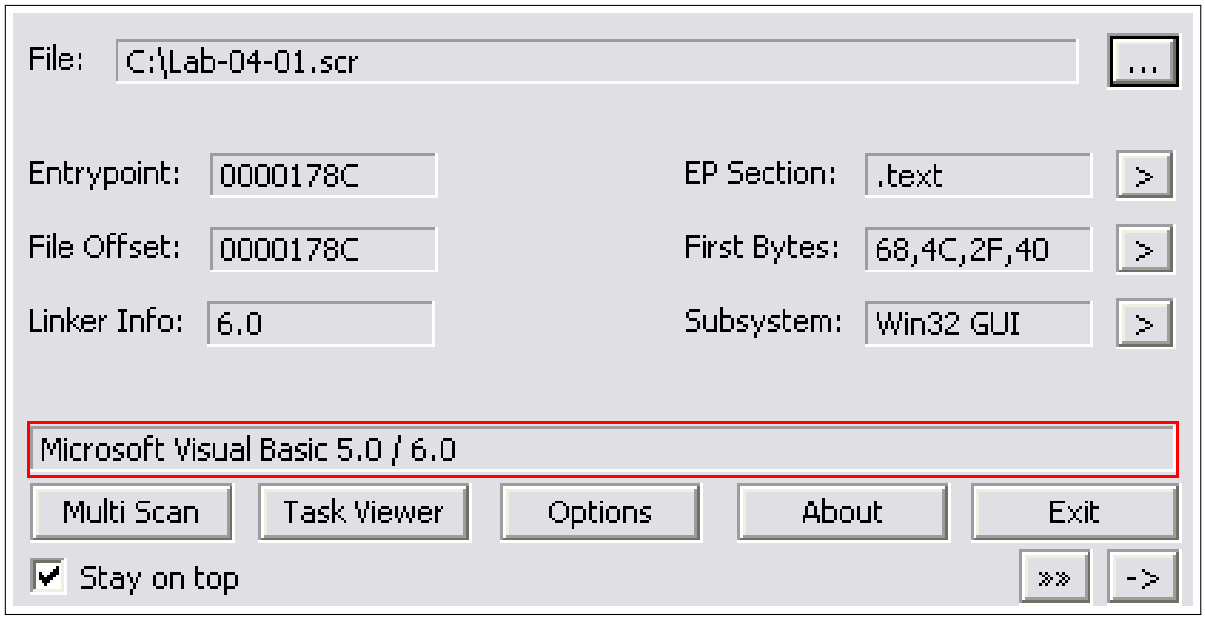


Figura 28 – Tela do *PEiD*, a qual indica que o arquivo foi desenvelopado

Próprio Autor

Depois de identificar que o arquivo estava empacotado e realizar o processo de desempacotamento, o código passou novamente pela ferramenta *PEiD*, como a imagem 28 mostra. Assim, é possível realizar a análise pelo *BinText*. Portanto, como mostra a figura 29 abaixo, foi identificado que a pessoa que programou esse *malware*, colocou-o na pasta documentos e *settings*. Também é possível visualizar o nome da pessoa.

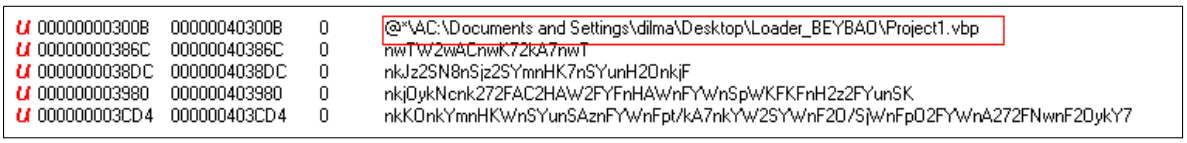


Figura 29 – Tela do *BinText* identificando pasta e nome da pessoa

Próprio Autor

Após esses procedimentos iniciais, a ferramenta utilizada foi o *OllyDGB*. Quando se realiza um *debugging* é possível navegar pelo código, além de ajudar na execução de trecho do mesmo, alterar valores e instruções em tempo de execução e permite analisar as *strings*. É interessante destacar a importância das *strings*, uma vez que podem indicar as ações realizadas pelo *malware*.

Portanto, antes de continuar a análise, é importante entender o que são essas *strings*. Uma *string* é uma sequência de caracteres, ou seja, é um conjunto de símbolos, os quais podem ser uma letra, um número, pontuação ou qualquer outro símbolo representado pelo teclado do computador. Assim, as *strings* podem indicar uma mensagem, uma *URL*, caminhos de arquivos e chaves no registro. No entanto, muitas vezes essas *strings* apare-

cem criptografadas, isto é, estão embaralhadas a fim de evitar a leitura e o entendimento delas. A figura 30 mostra um exemplo de *strings* criptografadas.

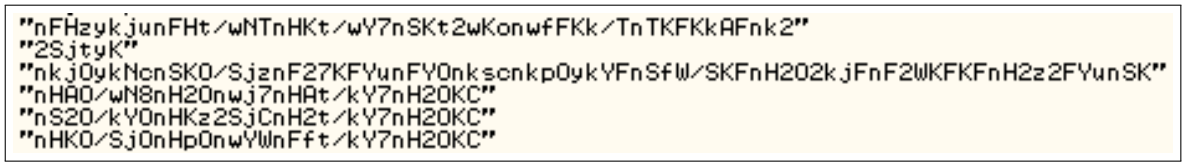


Figura 30 – Imagem indicando um exemplo de *Strings* criptografadas

Próprio Autor

Portanto, devido a importância das *strings*, nesse estudo é feito uma análise a fim de identificá-las. A ferramenta utilizada possui uma opção que consegue mostrar todas as *strings* do código analisado. Para realizar essa ação é só clicar com o botão direito para opção *search for* e depois ir em *all reference text strings*. Essa opção abrirá uma nova janela, a qual apresenta três colunas. A primeira coluna indica o endereço em que a *string* está, a segunda representa a instrução e a terceira indica um comentário da ferramenta. A imagem 31 mostra parte da janela aberta pela opção e as três colunas.

A ferramenta busca *strings* no local em que estão armazenadas, pois em alguma parte do segmento de memória há o armazenamento delas, e o local que estão sendo executadas, que representa a coluna *disassembly*. Em algumas dessas colunas há o comando MOV EDX, o qual indica onde a *string* está sendo utilizada.

Sabe-se que a instrução MOV se refere a uma *string* por conta do ponteiro, por exemplo, o valor 00.40.38.6C, como a imagem 31 mostra. Nesse endereço há uma *string* e na coluna da frente a mesma é apresentada. Para chegar à posição de memória onde a *string* está sendo utilizada, basta dar dois cliques na *string*. Nesse ponto, a ferramenta vai para a janela de código e mostra aonde a *string* está sendo utilizada, conforme pode ser visto na imagem 32.

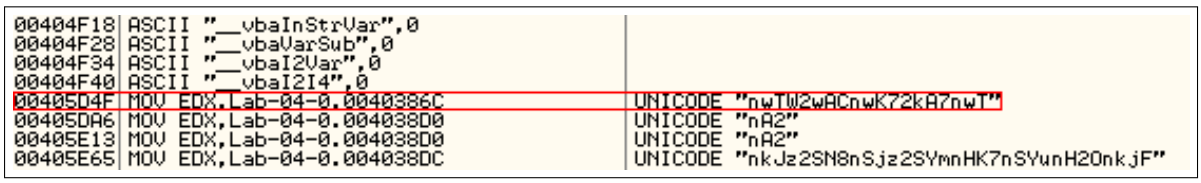


Figura 31 – Imagem da janela da opção que mostra as *strings* do programa analisado

Próprio Autor

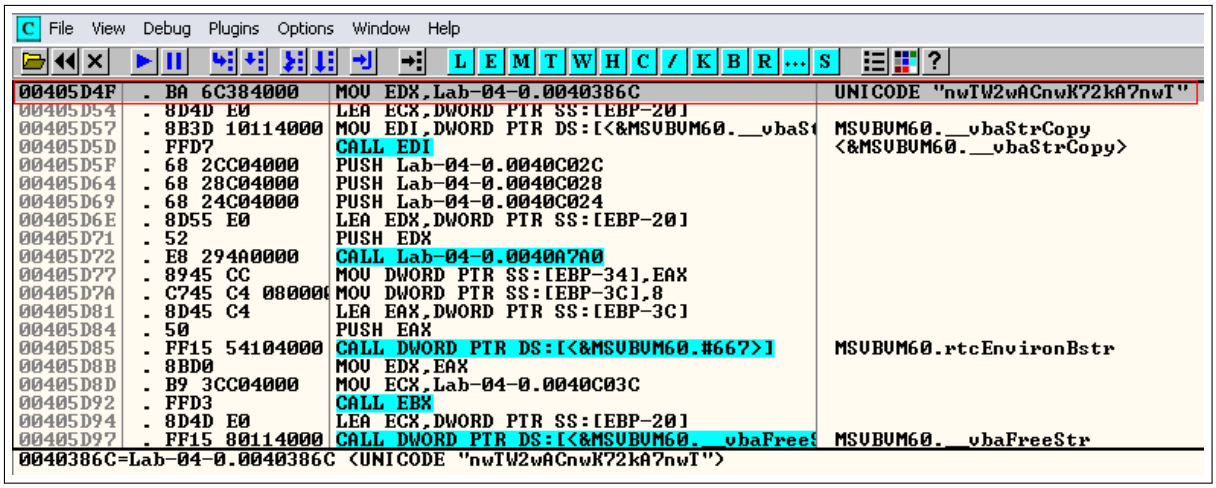


Figura 32 – Imagem que mostra aonde a *string* está sendo utilizada

Próprio Autor

É interessante destacar a importância do conjunto de instruções que se segue após a apresentação da *string*, já que é comum ter no código a função que realiza a descriptografia das *strings*. Assim, para conseguir identificar essa função, é importante verificar se há um padrão no código analisado.

As imagens 33 e 34 abaixo mostram a sequência de instruções após a apresentação de duas *strings*. É interessante destacar que ambas possuem uma chamada CALL com o registrador ESI, o qual também pode ser EDI, quatro instruções PUSH e uma chamada CALL com o mesmo endereço 40A7A0.

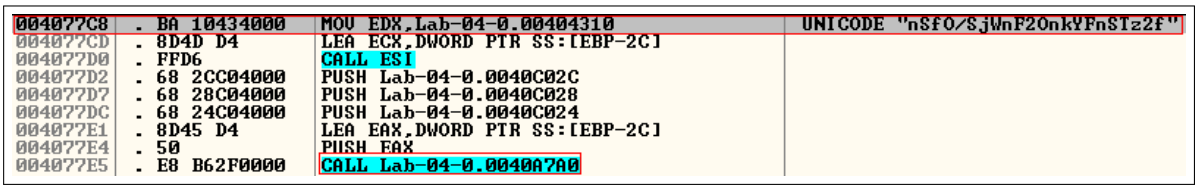


Figura 33 – Imagem que mostra o trecho de código após a apresentação de uma *string*

Próprio Autor

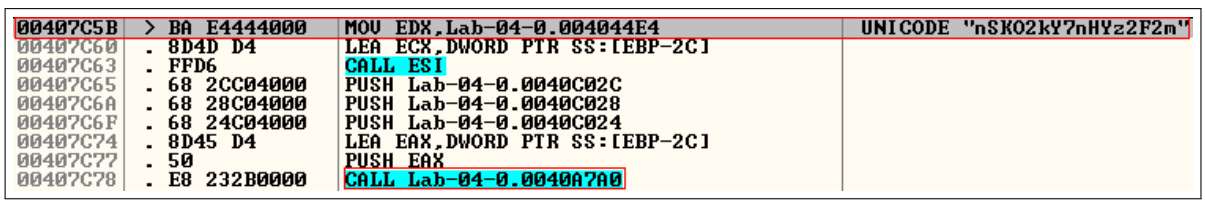


Figura 34 – Imagem que mostra o trecho de código após a apresentação de outra *string*

Próprio Autor

O fato deste padrão de código sempre se repetir torna esse código muito suspeito, já que a *string* possui a função de descriptografia logo após a sua chamada. Desta forma, pode-se concluir que essa função é a que faz a descriptografia da *string*. A mesma função está sendo utilizada para descriptografar todas as *strings*.

Como o código sempre realiza a chamada CALL para o mesmo endereço será colocado um *breakpoint* nele. Um *breakpoint* é um ponto de parada, ou seja, quando executar o código, o mesmo irá executar até esse ponto. Para colocar um *breakpoint* é só apertar a tecla F2.

Após colocar o *breakpoint* é possível executar o código. É importante observar, na janela de registradores, o padrão de *string* executado. Como não se sabe se a execução do código será muito demorada, então a opção de executar linha a linha será descartada, pois pode ter muitas instruções, *loops* e chamadas recursivas. Neste ponto, o importante é o retorno da função, já que ele mostrará a *string* descriptografada.

Portanto, para executar a opção inteira sem interrupções, a ferramenta possui a opção *execute till return* em *debug*. Após a execução será utilizado a tecla F7 para executar linha a linha até chegar ao código que chama a função mencionada. No final, chega-se ao trecho de código destacado nas imagens anteriores. Nesse ponto podemos visualizar o resultado do retorno. Normalmente, o resultado aparece no registrador EAX. Assim sendo, visualizando o EAX aparecerá a *string* "APPDATA". A imagem 35 mostra o padrão de código e o retorno da função no registrador com a *string* "APPDATA".

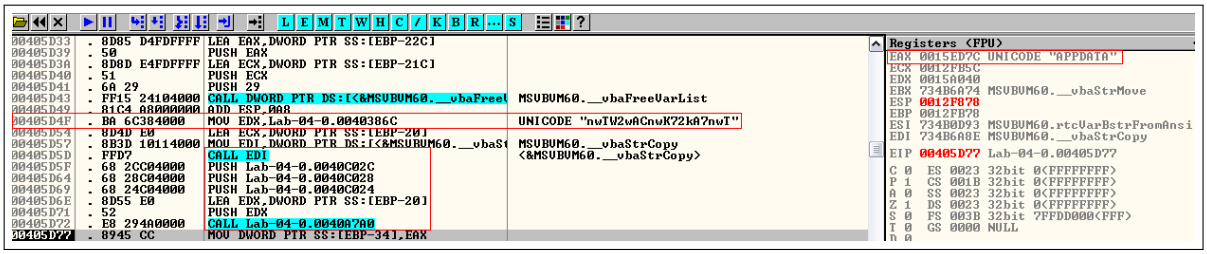


Figura 35 – Imagem que mostra a *string* criptografada, o padrão de código analisado e a *string* descriptografada no registrador EAX

Próprio Autor

Após identificar a função que realiza a descriptografia e verificar o retorno, é possível que outras *strings* sejam analisadas. Como a mesma função realiza a descriptografia de todas as outras *strings*, foi selecionado um trecho de código em que possui uma *string*. Assim, foi colocado um *breakpoint* no endereço 405D4F e retirado o ponto de parada que estava em 405A7A0. Após colocar esse novo *breakpoint* o código foi executado.

Ao parar no *breakpoint*, o valor da *string* utilizada foi alterado para o valor de outra *string*. Para realizar essa mudança é só clicar com o botão direito em cima da *string* e selecionar a opção *assemble*. Essa opção abrirá uma janela com a instrução MOV, como a imagem 36 mostra.

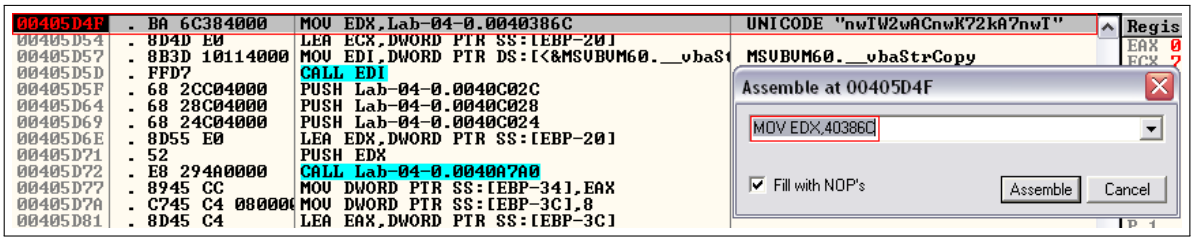


Figura 36 – Imagem que mostra a janela da opção para alterar o valor das *strings*

Próprio Autor

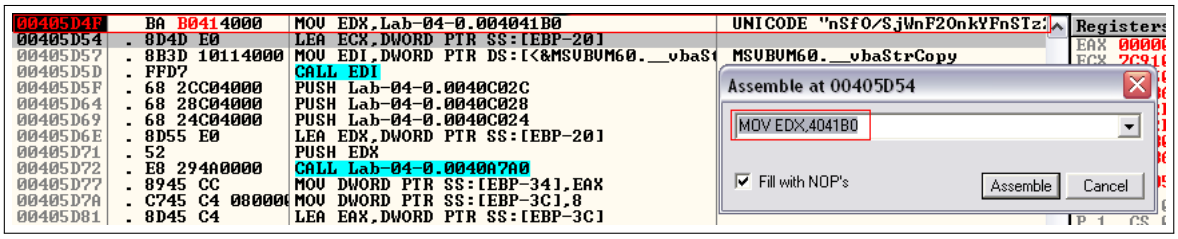


Figura 37 – Imagem mostrando a alteração dos valores das *strings*

Próprio Autor

Nessa análise, o valor 40486C foi alterado para 4041B0 como a imagem 37 mostra. Ao realizar a alteração, é possível continuar a análise. Para prosseguir, foi utilizada a opção F8 para seguir linha a linha até a instrução MOV após a chamada CALL. Ao chegar na instrução MOV, foi visto que a *string* analisada foi descriptografada. A imagem 38 mostra a *string* antes de passar pela instrução CALL e a imagem 39 mostra a *string* após passar pela função que realiza a descriptografia.

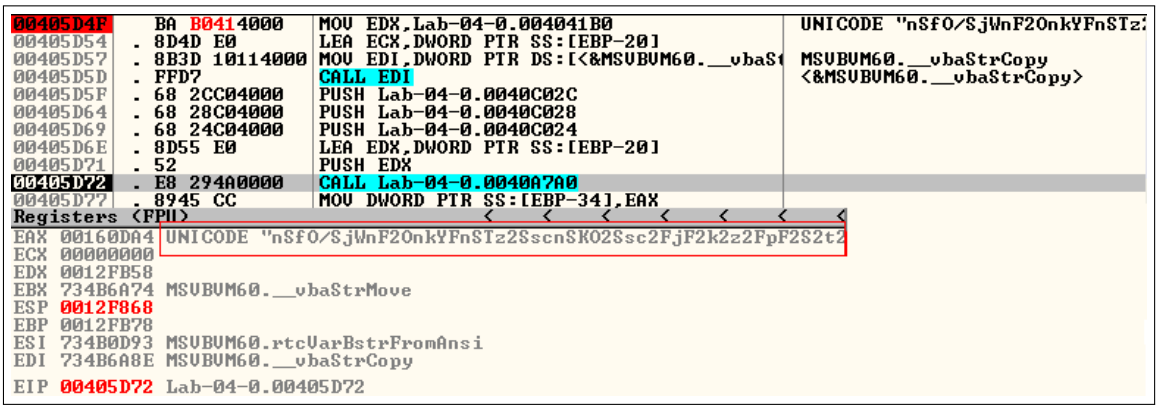


Figura 38 – Imagem que mostra a *string* analisada criptografada

Próprio Autor

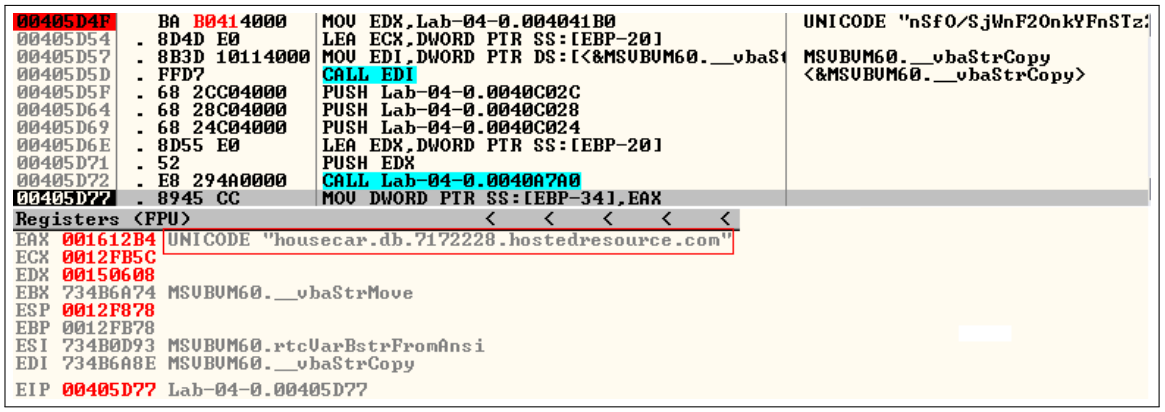


Figura 39 – Imagem que mostra a *string* analisada descriptografada

Próprio Autor

A imagem 40 mostra outras *strings* descriptografadas do arquivo estudado. O código tenta obter dados como o ID e a senha do usuário. O *malware* também tenta obter acesso ao *root* do computador e por isso imagina-se que o mesmo quer ter acesso privilegiado. Portanto, ao tentar ter acesso de administrador, acessar a rede e também obter informações confidencias, o código pode-se classificar como um *spyware*.

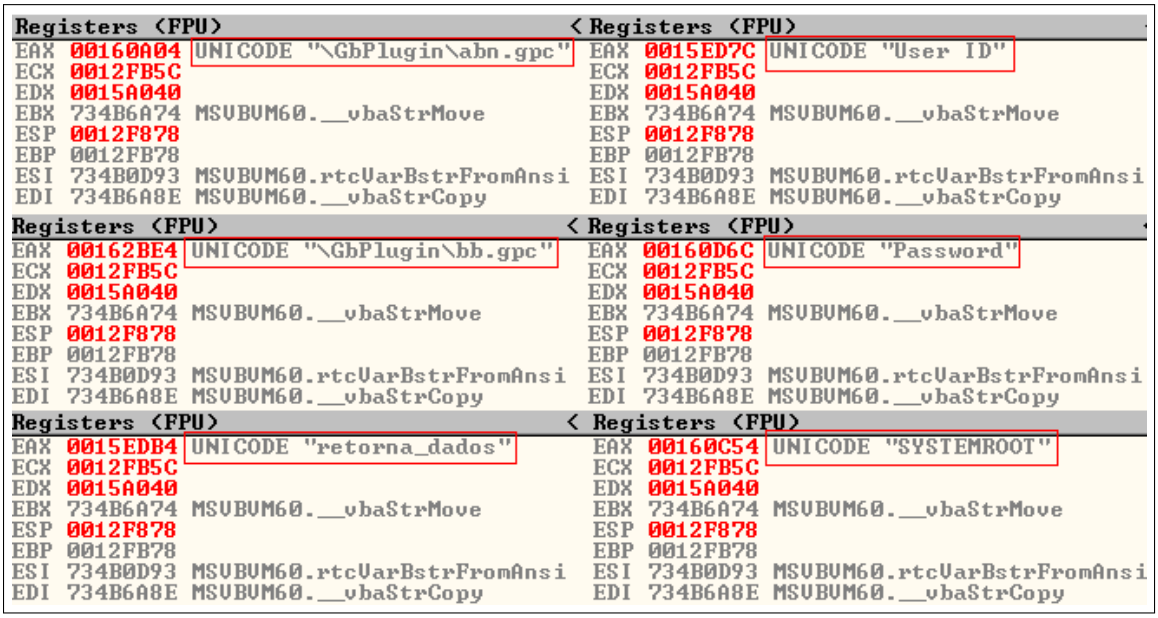


Figura 40 – Imagem que mostra outras *strings* analisadas

Próprio Autor

Segundo Stallings e Brown (2015), um *spyware* é um *software* que obtém informações confidenciais e envia para outro sistema que monitora o teclado, dados da tela e tráfego da rede, ou faz a digitalização de arquivos de um sistema para informações sensíveis.

5.4 Arquivo Executável criando Processos

O arquivo desse estudo é um executável e as ferramentas utilizadas foram *Process Explorer* e *Process Monitor*. Assim, para iniciar a análise, foram abertos os programas e inicializado o *Process Monitor*. Para inicializar essa ferramenta é só ir em *file* e deixar a opção "capturar eventos"marcada. Após a inicialização da ferramenta, pode-se executar o código malicioso. A imagem 41 mostra o arquivo aberto no *Process Monitor*.

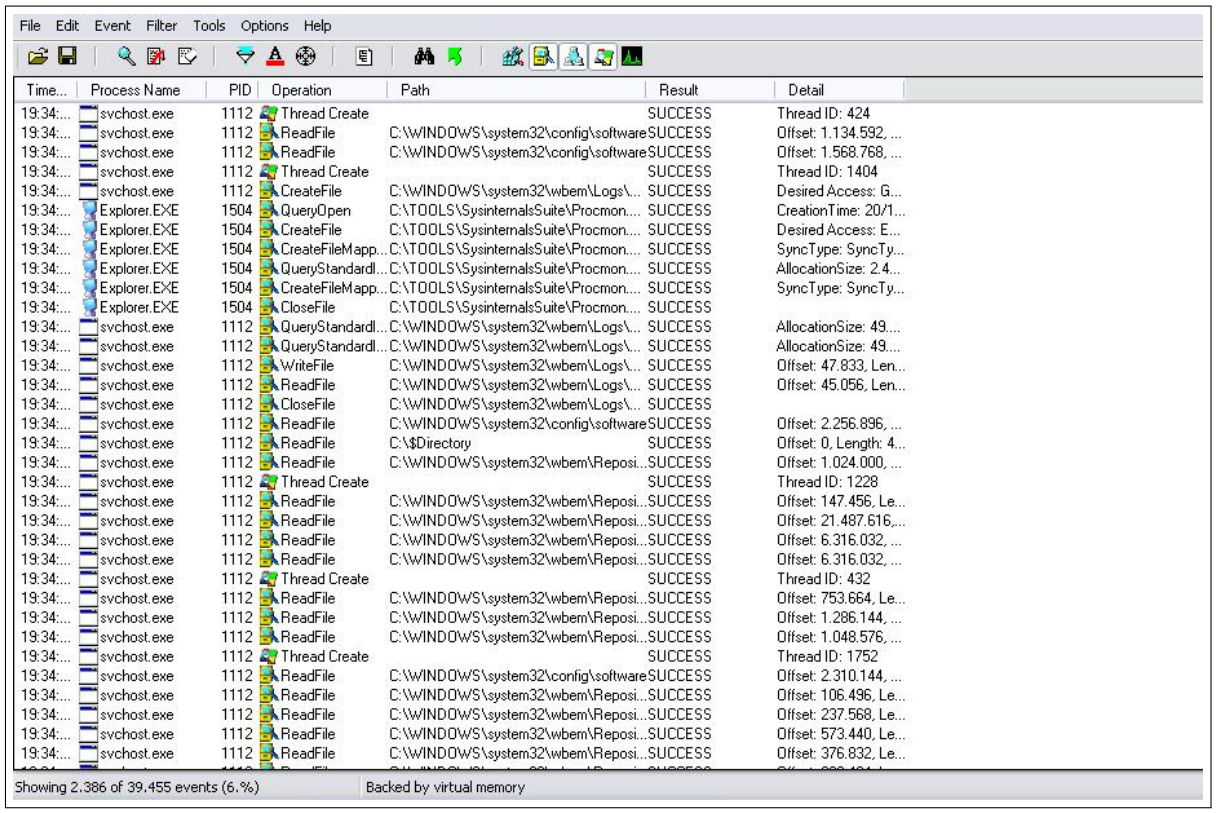


Figura 41 – Tela do *ProcMon* com o arquivo analisado aberto

Próprio Autor

Depois de capturar os eventos, é importante analisar cada um deles. Assim, como teve muitos eventos capturados, foi filtrado para mostrar somente aqueles eventos relacionados ao código analisado. Então, foi filtrado para parecer aqueles eventos relacionados ao nome do arquivo.

Durante a análise, foi observado que o arquivo faz várias chamadas as *APIs* do sistema. No entanto, percebeu-se que, após certo ponto, o mesmo cria um processo na pasta *System32* do *Windows* com o nome de *svchost.exe*. Após a criação do processo, o mesmo fecha um arquivo, fecha uma *thread* e fecha o processo inicial, porém não finaliza o processo criado anteriormente. A imagem 42 mostra essas ações.

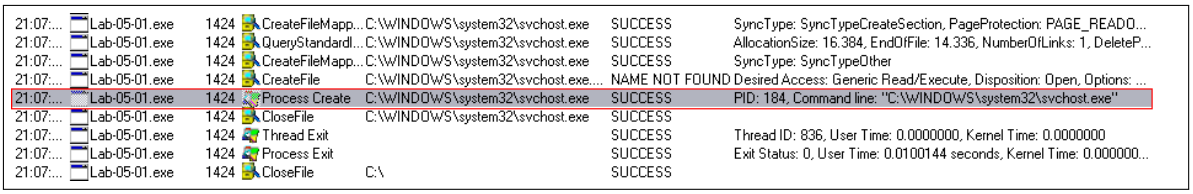


Figura 42 – Tela do *Process Monitor* que mostra o arquivo criando um processo

Próprio Autor

É importante destacar que *svchost* é um espaço, o qual o sistema utiliza para executar suas aplicações. Esse processo possui um "pai"e quando chamado, o mesmo possui parâmetros. No entanto, ao verificar o *Process Explorer*, o processo *svchost.exe*, que está sendo executado, não possui um "pai", não tem parâmetros e possui o mesmo *PID* do processo visto no *Process Monitor*. Assim, como o *PID* é igual, pode-se afirmar que é o mesmo processo. As figuras 42 e 43 mostram o processo e o *PID*.

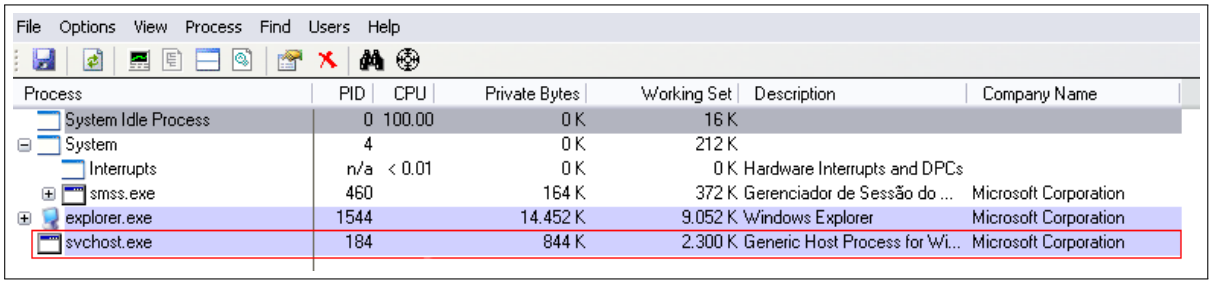


Figura 43 – Tela do *Process Explorer* com o processo *svchost* operando na máquina

Próprio Autor

Na ferramenta *Process Explorer*, ao ir em propriedades, para obter algumas informações sobre o processo, foi observado que o arquivo mostrava que era do sistema. Para confirmar essa informação, foi pedido uma verificação. Após esse pedido, foi visto que o mesmo é um arquivo original do *Windows*. As imagens 44 e 45 mostram essas informações.

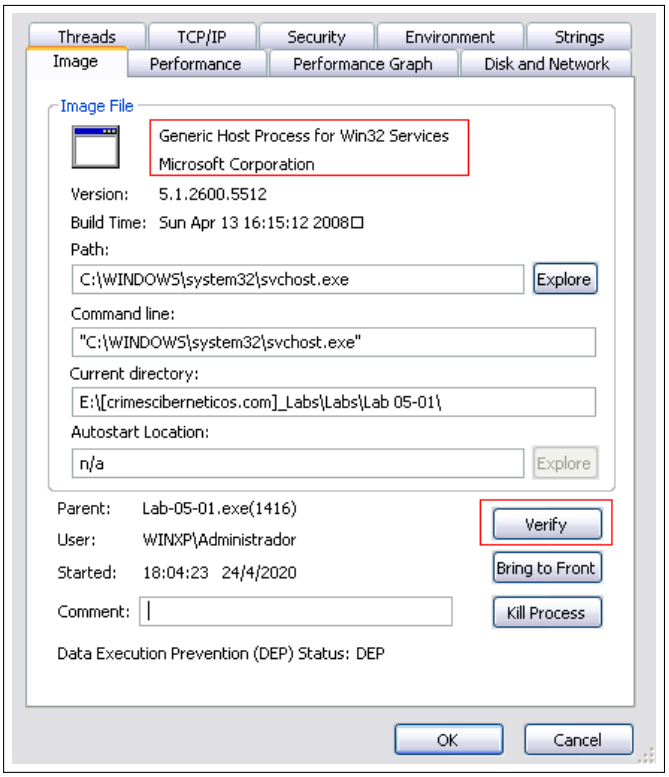


Figura 44 – Tela do *Process Explorer* com a informação que o arquivo era do sistema

Próprio Autor

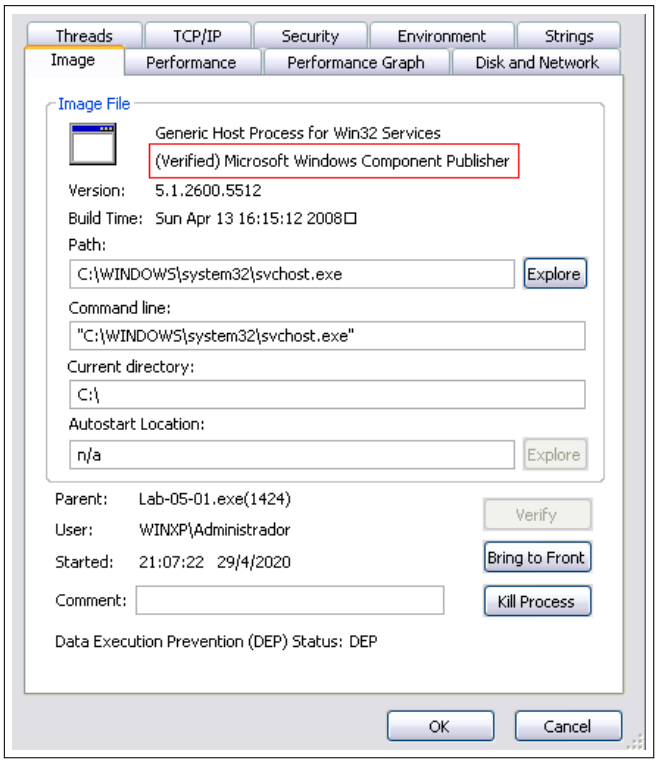


Figura 45 – Tela do *Process Explorer* após a verificação solicitada

Próprio Autor

É interessante lembrar que quando ocorre uma verificação de um arquivo, é analisado o disco rígido, porém não é examinado a memória. Assim, para ter certeza que o processo é do sistema, foi aberto outra tela de propriedades e comparado as informações contidas no disco rígido e as informações da memória. A figura 46 mostra as informações do disco rígido, e 47 as informações contidas na memória.

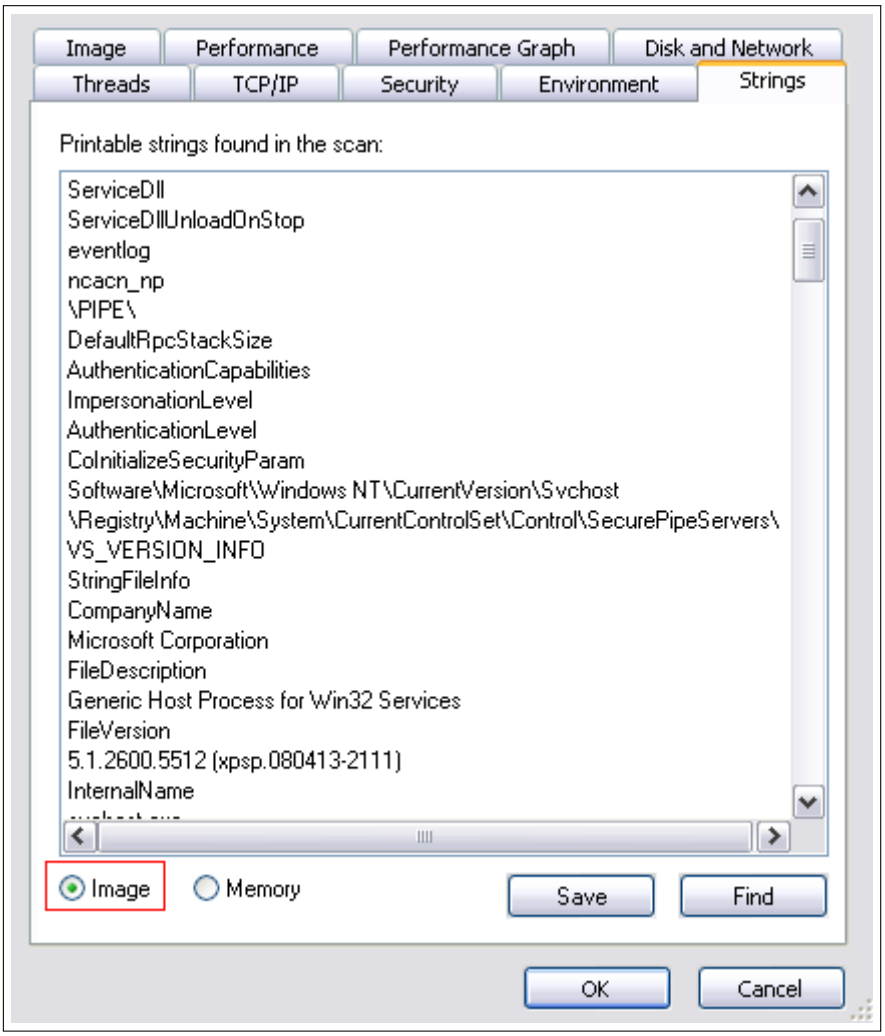


Figura 46 – Tela que mostra as informações no disco rígido

Próprio Autor

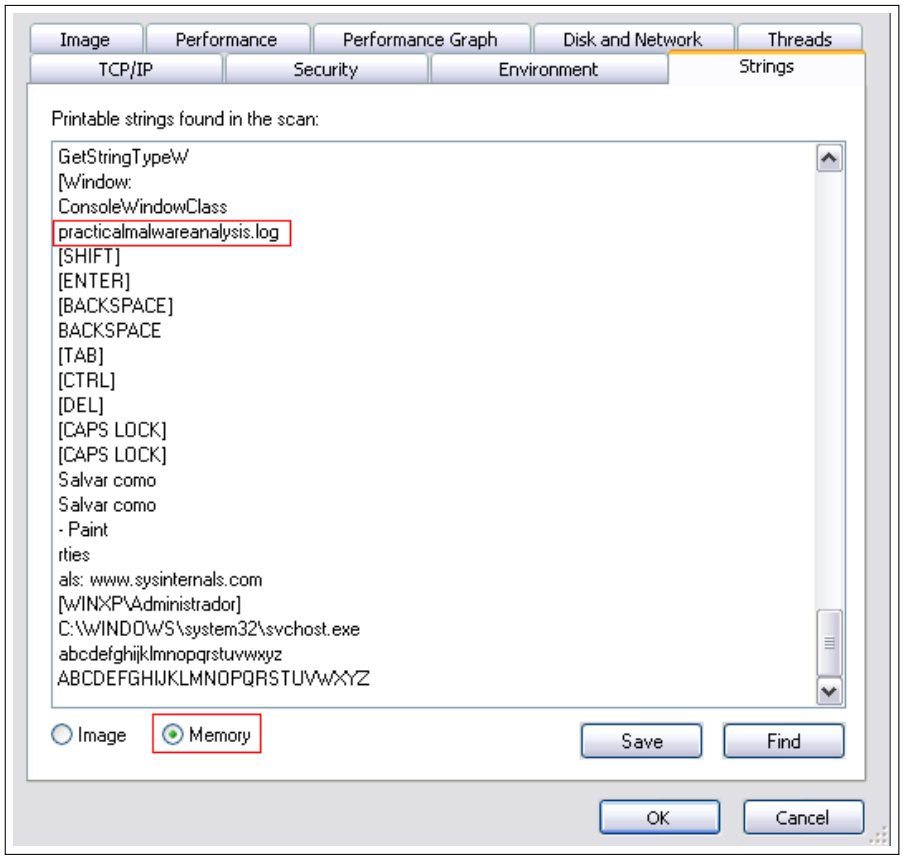


Figura 47 – Tela que mostra as informações da memória

Próprio Autor

Como visto nas imagens 46 e 47, o conteúdo da memória é diferente do que se tem no disco, ou seja, as informações na memória foge do padrão do disco. Também é importante destacar que há um arquivo com extensão *log*. Esse tipo de arquivo, segundo Barros (2018), registra eventos relacionados ao sistema operacional, aplicativos que estão em execução e dispositivos de rede. Assim, depois dessa comparação, também foi visto que o processo ainda estava ativo. Portanto, para saber o que o mesmo estava realizando na máquina, foi selecionado, no *Process Monitor*, os eventos relacionados ao PID do processo.

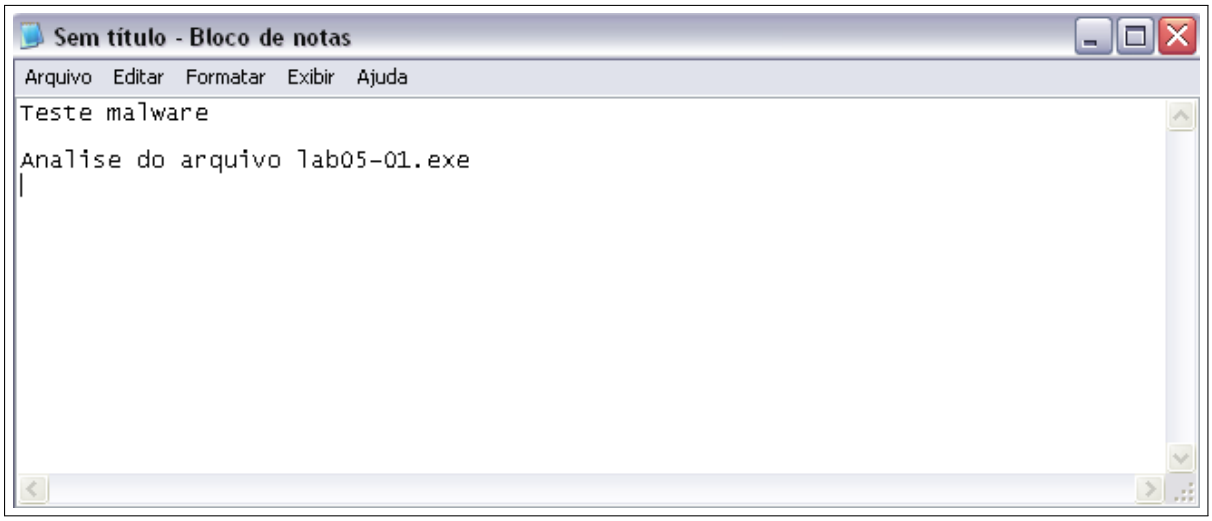


Figura 48 – Tela do bloco de notas realizando o teste

Próprio Autor

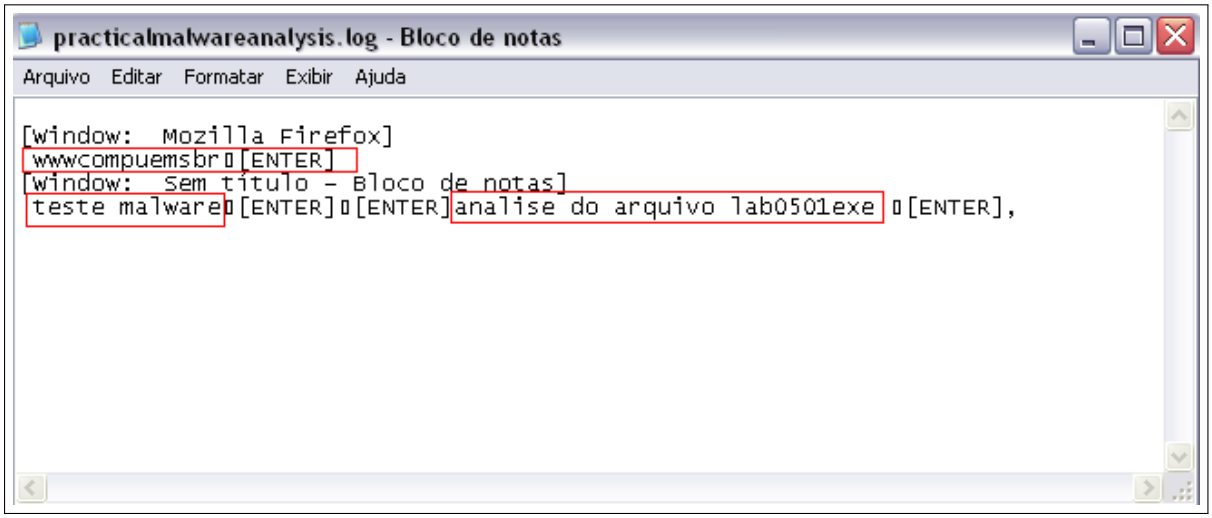


Figura 49 – Tela do bloco de notas com o arquivo *log* aberto

Próprio Autor

O código analisado tem comportamento parecido com o *malware Keylogger*, ou seja, o mesmo fica disfarçado de arquivo de sistema para capturar tudo que o usuário da máquina atingida digita. Para realizar um teste, foi aberto o navegador e digitado o *site www.comp.uems.br* e depois foi aberto um arquivo no bloco de notas. Após isso, foi visto no arquivo *praticamalwareanalysis.log*, as atividades realizadas. As imagens 48 e 49 apresentam essas ações.

5.5 Malware Acessando Site Externo pela Rede

Esse estudo envolve um arquivo executável, o qual foi analisado pela ferramenta *FakeNet*. Assim, antes de começar a análise, é importante verificar se na máquina virtual a configuração de rede está com a opção *host-only*. Essa opção faz com que a máquina virtual não acesse uma rede externa a ela. Após essa verificação, inicializar o *FakeNet* e depois executar o *malware*.

A imagem 50 mostra as atividades que o código realiza. O *malware* faz uma pesquisa ao *DNS*, ou seja, o mesmo tenta encontrar o dominio *dl.dropbox.com*. Após isso, o código faz uma requisição na porta 80 e tenta trazer para a máquina os métodos *metodoS.swf, metodoD.swf, metodoL.swf* e *metodoAux.swf*. Se a página procurada estivesse ativa, então seria possível realizar o *download* desses arquivos.

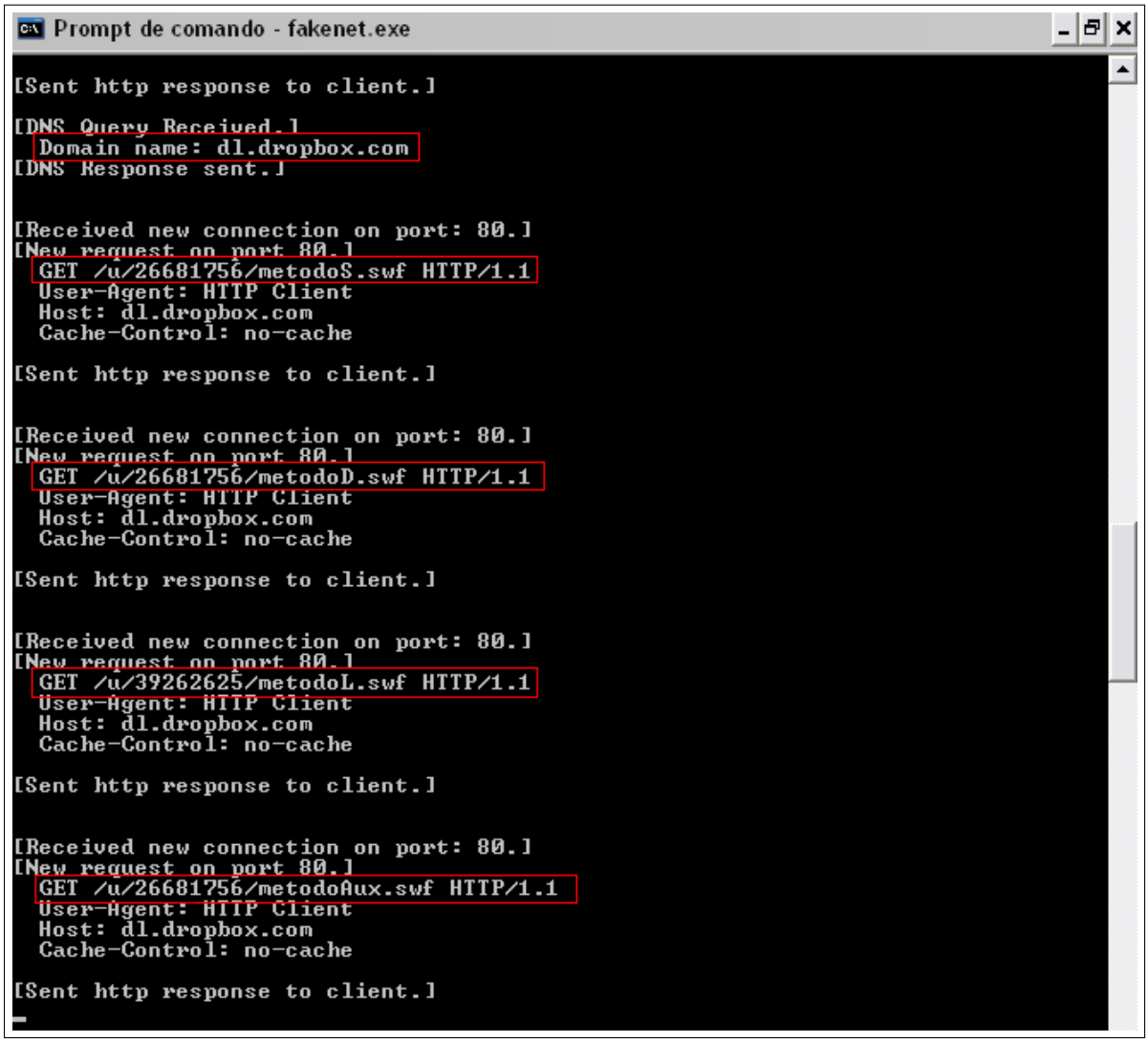


Figura 50 – Tela do *FakeNet* com as atividades do *malware*

Próprio Autor

Assim, como o código tenta trazer arquivos para a máquina atingida, o programa pode se encaixar na categoria *Downloader*. De acordo com Stallings e Brown (2015), um *Downloader* é um código, o qual instala outros itens em uma máquina que está sob ataque. Geralmente está incluído no código do *malware* inserido pela primeira vez em um sistema comprometido para então importar um pacote maior de *malware*.

5.6 Arquivo disfarçado de executável

Essa seção envolve o estudo de um arquivo executável, o qual passou pelas seguintes ferramentas: *PEiD, Exeinfo PE, Delphi Decompiler* e *Delphi 7*. Para iniciar análise, a primeira ferramenta utilizada foi *PEiD* para obter informações sobre o formato do arquivo e qual compilador foi usado pelo código. A imagem 51 abaixo mostra essas informações.

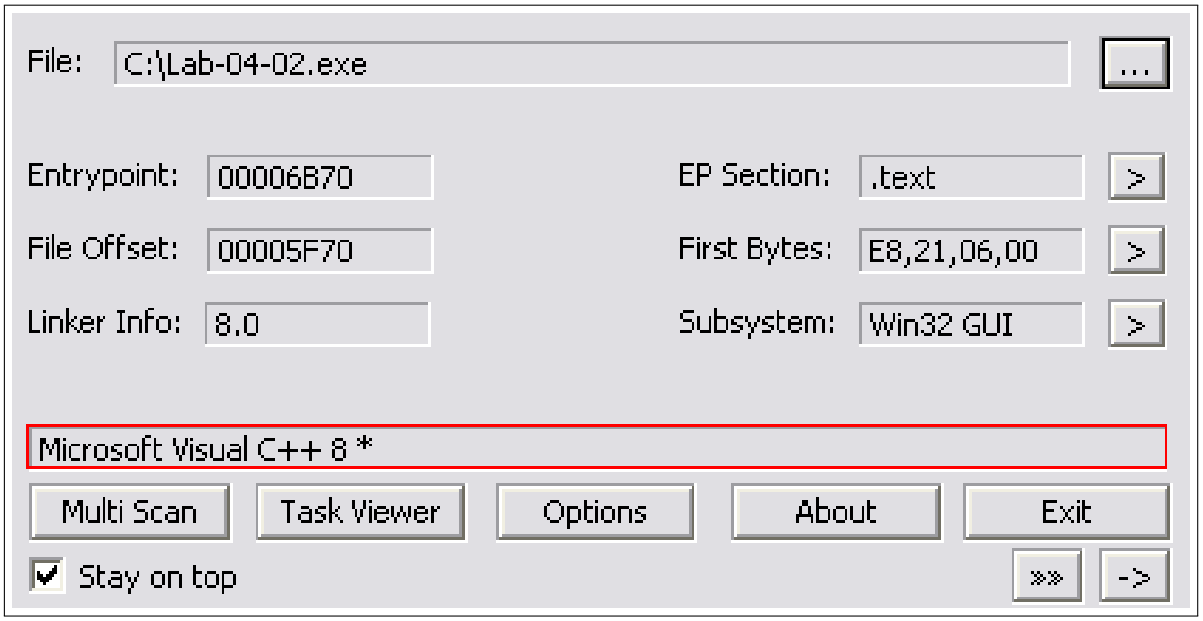


Figura 51 – Tela do *PEiD* com o arquivo analisado aberto

Próprio Autor

Após utilizar o *PEiD*, foi usado o *Exeinfo PE*. Portanto, quando aberto pela ferramenta, a mesma sugere que o arquivo seja renomeado para extensão *.CAB*. A imagem 52 mostra o arquivo aberto pelo programa e essa informação.

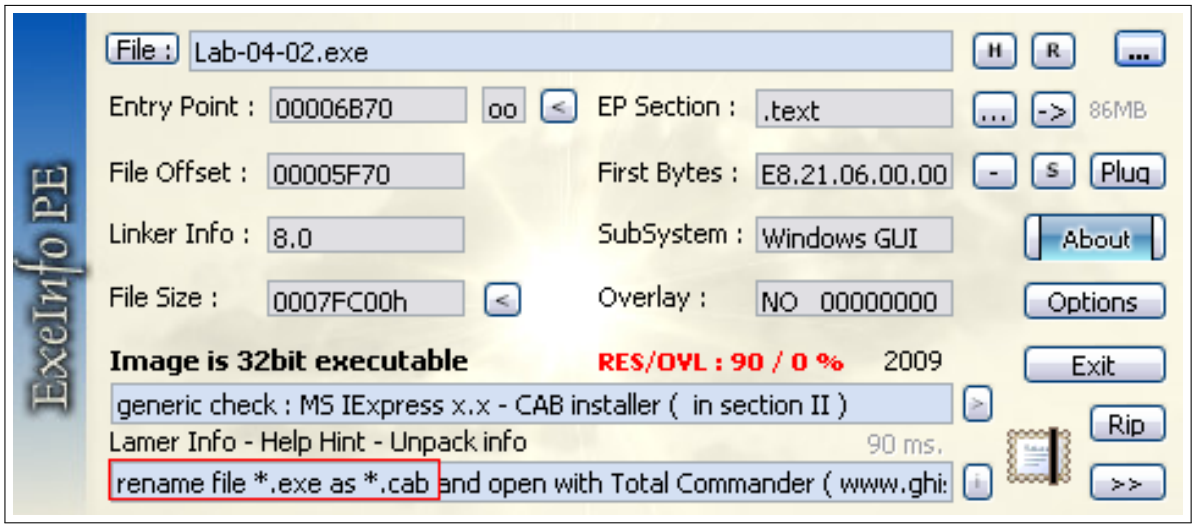


Figura 52 – Tela do *Exeinfo PE* com o arquivo analisado aberto

Próprio Autor

Portanto, o arquivo foi renomeado para a extensão indicada pela ferramenta. É importante destacar que arquivos com extensão *.CAB* apresentam dados compactados. Assim, após renomear o arquivo analisado, o mesmo apresentou ser um arquivo compactado. Depois de descompactá-lo, o mesmo possuía mais dois arquivos com extensão *.EXE*, os quais também passaram pela ferramenta *PEiD* como mostram as imagens 53 e 54 abaixo.

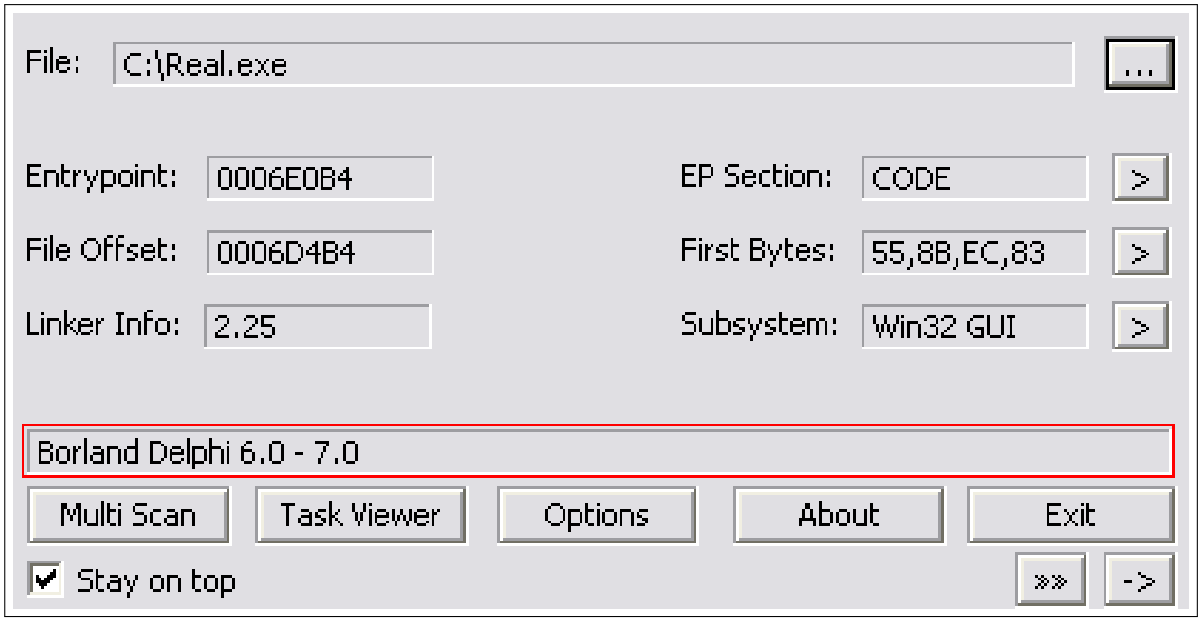


Figura 53 – Tela do *Peid* com o arquivo analisado aberto

Próprio Autor

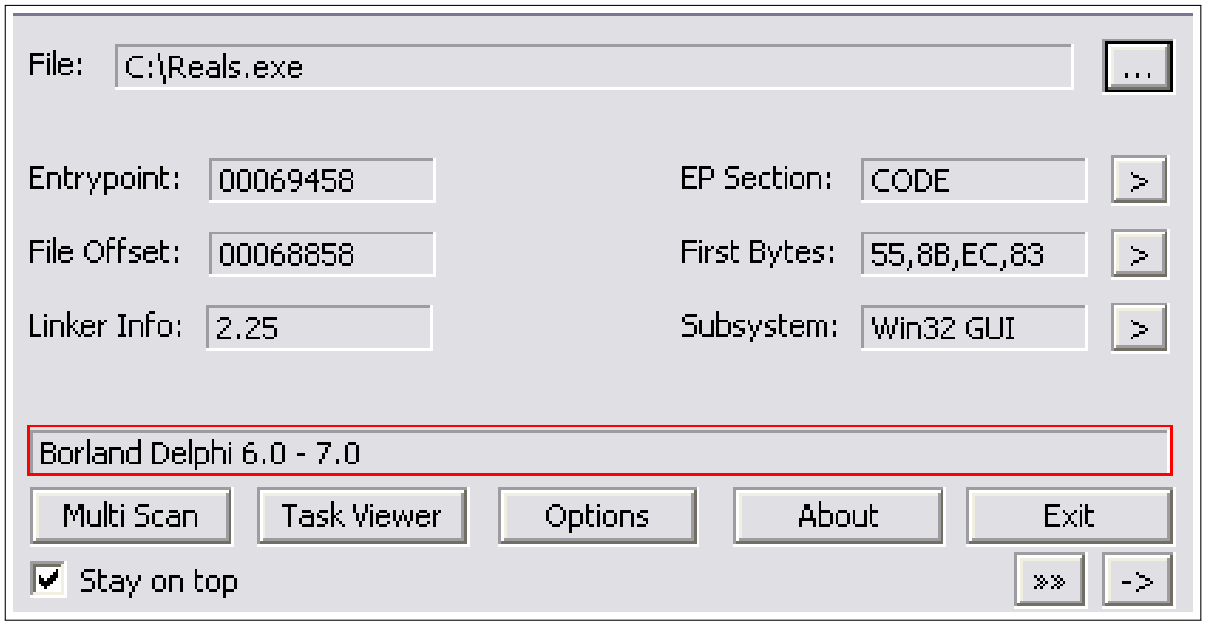


Figura 54 – Tela do*Peid* com o arquivo analisado aberto

Próprio Autor

É interessante destacar que o compilador que as imagens 53 e 54 mostram é o *Borland Delphi*. No entanto, foi visto anteriormente que o compilador utilizado era *Microsoft Visual C++*. Assim, imagina-se que o desenvolvedor do programa malicioso queria evitar uma possível análise tentando enganar os analisadores.

Como os códigos foram desenvolvidos na linguagem *Delphi*, é possível abrir esses arquivos. Portanto, para realizar essas atividades foram utilizadas as ferramentas *Delphi Decompiler* e *Delphi 7*.

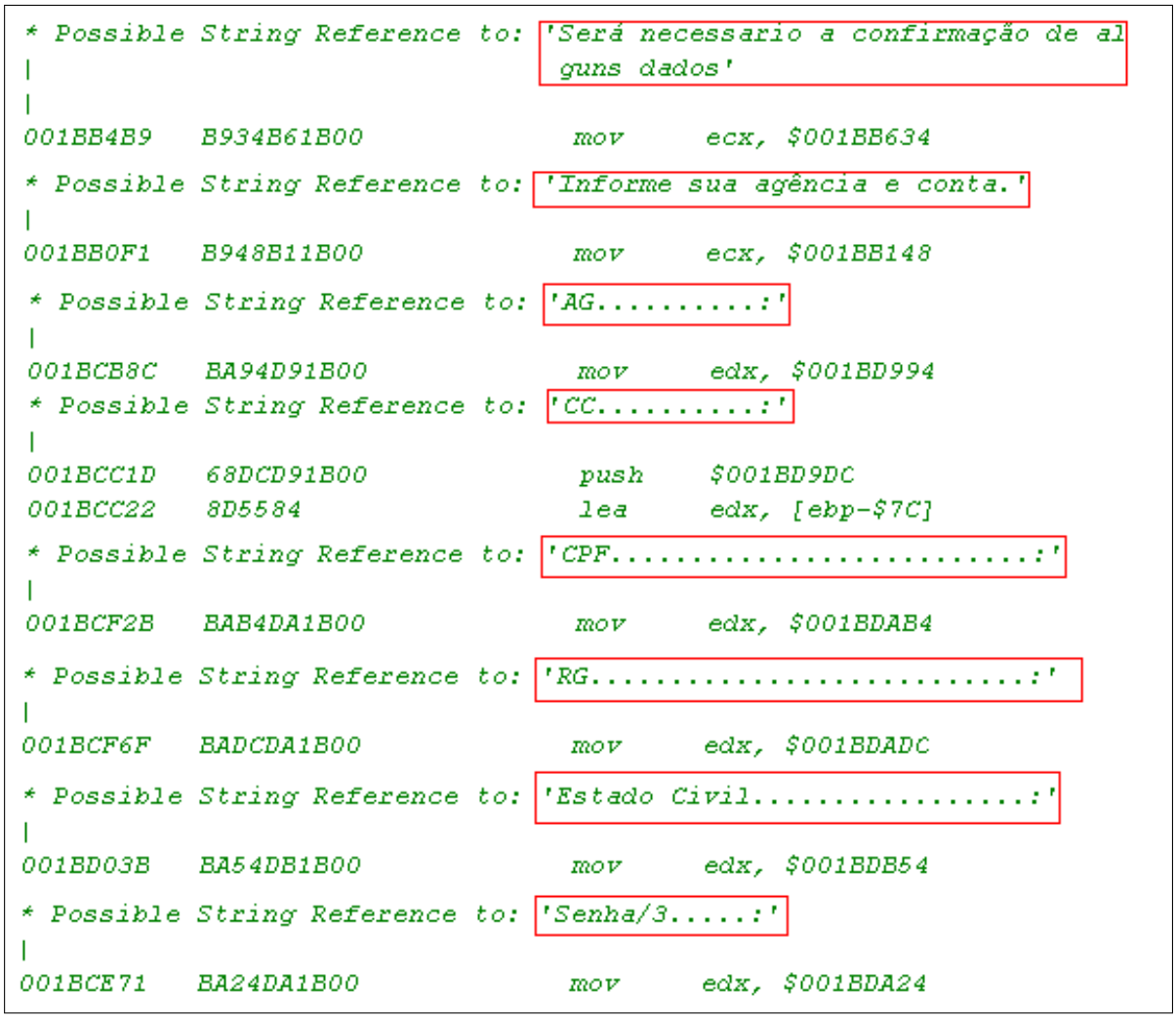


Figura 55 – Tela do *Delphi 7* com o primeiro arquivo analisado aberto

Próprio Autor

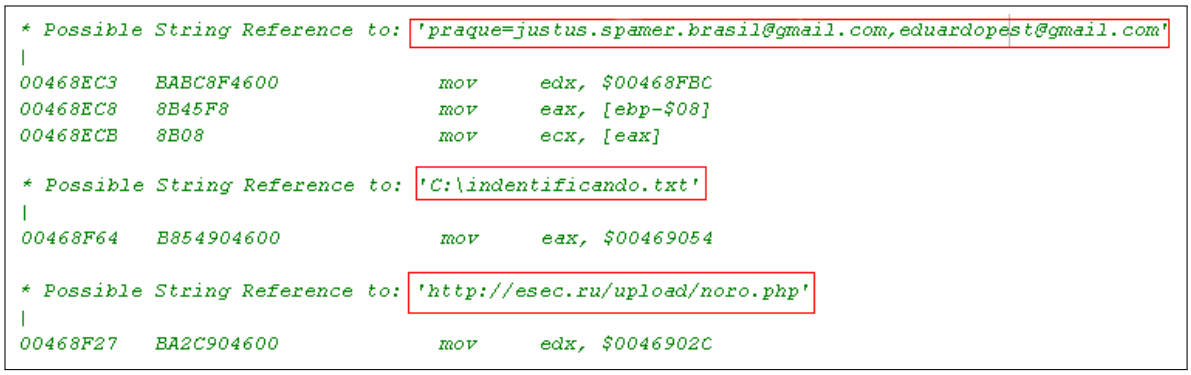


Figura 56 – Tela do *Delphi 7* com o segundo arquivo analisado aberto

Próprio Autor

O arquivo *real.exe* apresentava *strings* referentes ao RG, CPF, estado civil e senha, além de outras, como mostra a imagem 55. Já o segundo arquivo, *reals.exe*, possuia

strings indicando um endereço de e-mail, uma *url* e um arquivo texto, como a imagem 56 apresenta. Também foi visto na ferramenta *Delphi Decompiler* um texto do arquivo *reals.exe*. A imagem 57 mostra o texto identificado.

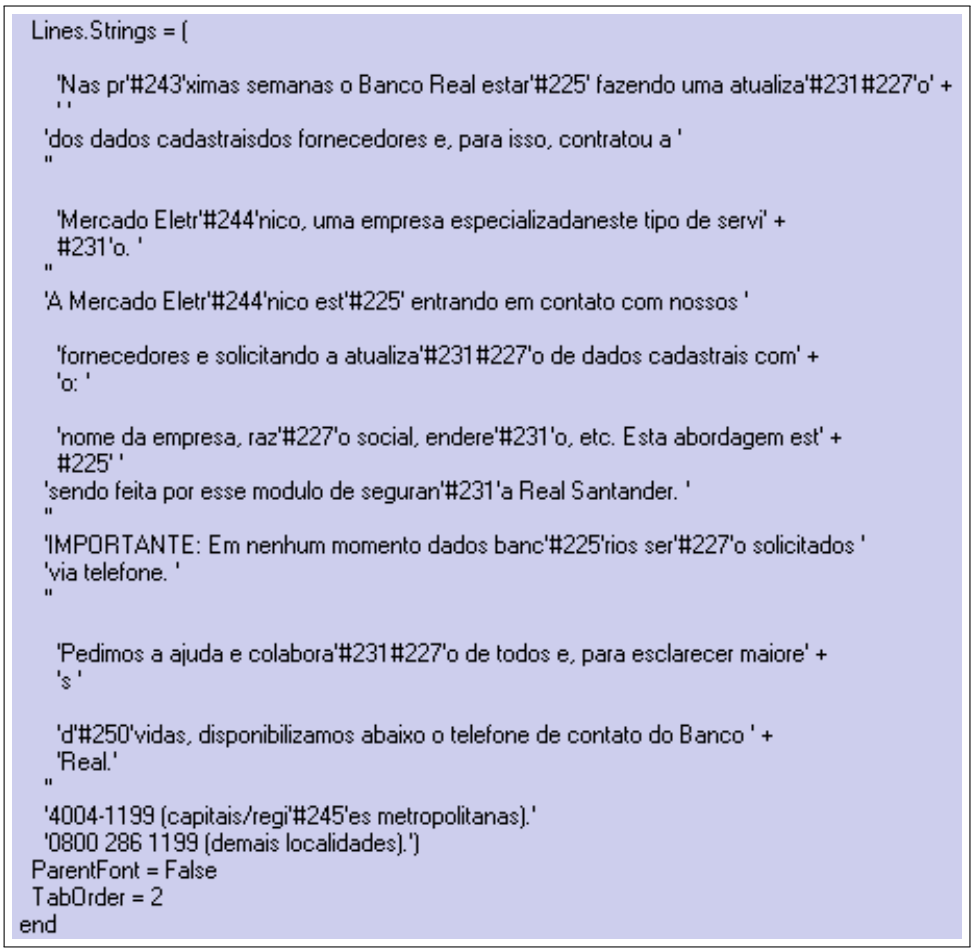


Figura 57 – Tela do *Delphi Decompiler* com o segundo arquivo analisado aberto

Próprio Autor

Depois de identificar as *strings*, foi visto também que o código analisado apresentava uma estrutura de um formulário. Assim, depois de realizar a análise, os arquivos foram executados para ver o que os mesmos realizavam na máquina. As imagens 58, 59 e 60 representam o formulário que o código possui. É interessante destacar que as *strings* observadas na imagem 57 aparecem na tela inicial que o *malware* criou. A imagem 58 apresenta as *strings* utilizadas no formulário.

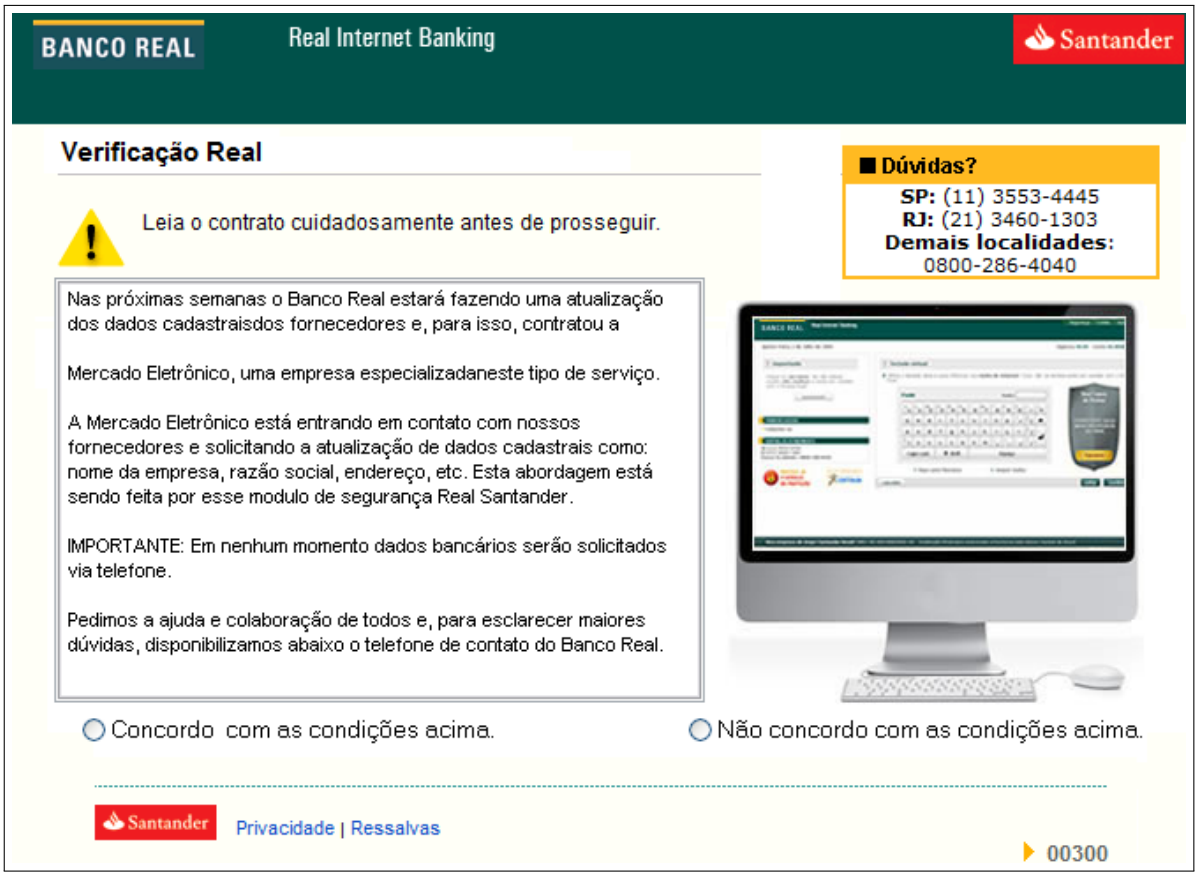


Figura 58 – Tela inicial do formulário feito pelo *malware*

Próprio Autor

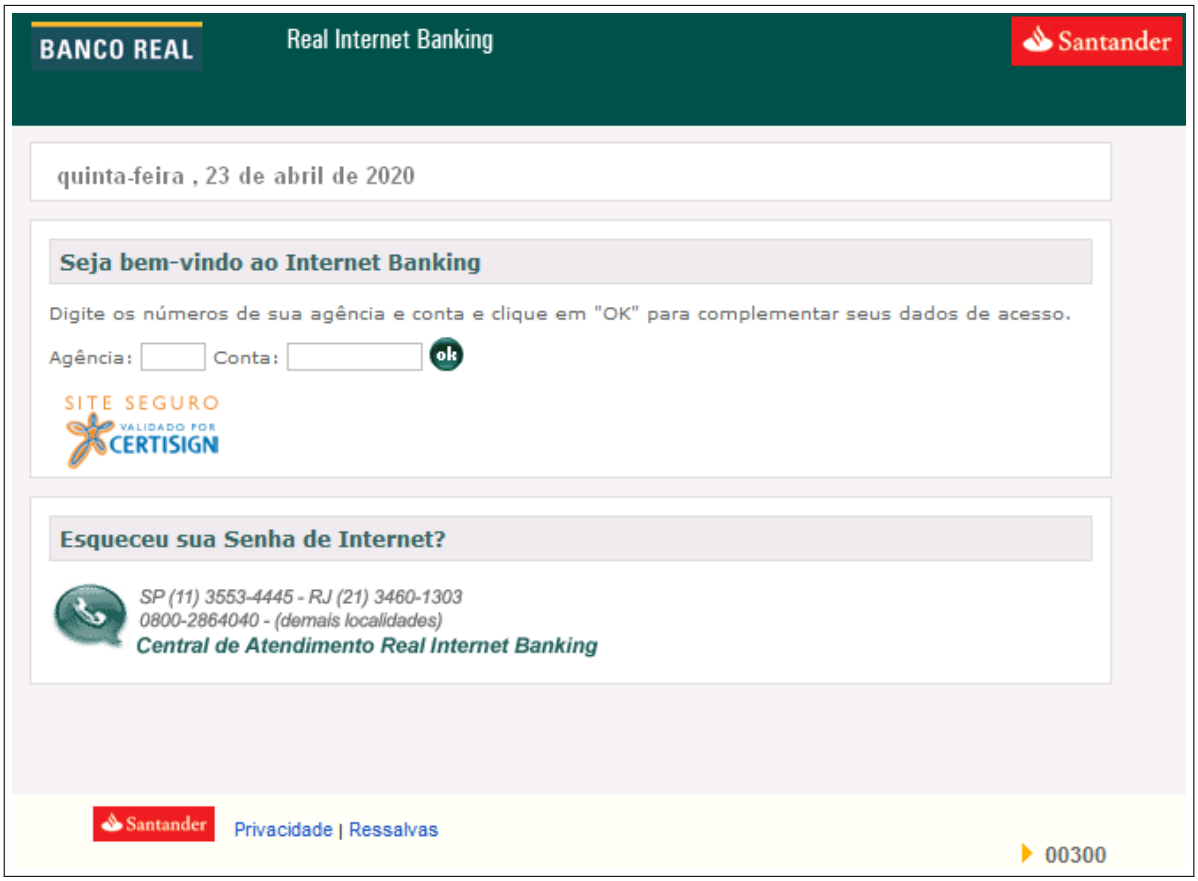


Figura 59 – Segunda tela do formulário feito pelo *malware*

Próprio Autor

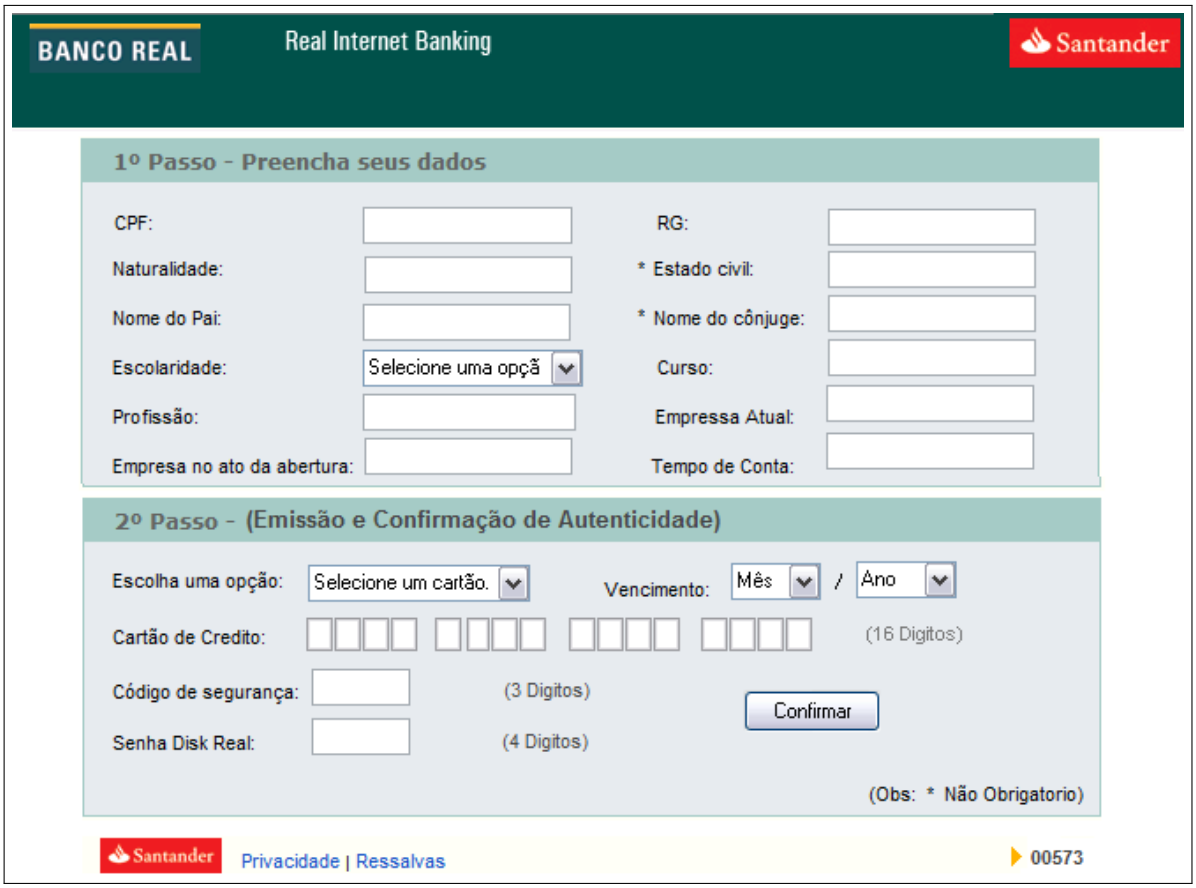


Figura 60 – Terceira tela do formulário feito pelo *malware*

Próprio Autor

Para realizar um teste e entender o que o *malware* faz na máquina foi preenchido o formulário criado pelo mesmo. As imagens 61 e 62 mostram essas informações. Após realizar esse procedimento foi aberto o arquivo texto que o código malicioso cria. Ao abrir o arquivo foi visto que as informações digitadas no formulário apareciam nesse arquivo. Portanto, acredita-se que o *malware* copiava as informações nesse arquivo texto para enviar nos endereços de *e-mails* vistos. A imagem 63 indica o arquivo texto criado pelo código.

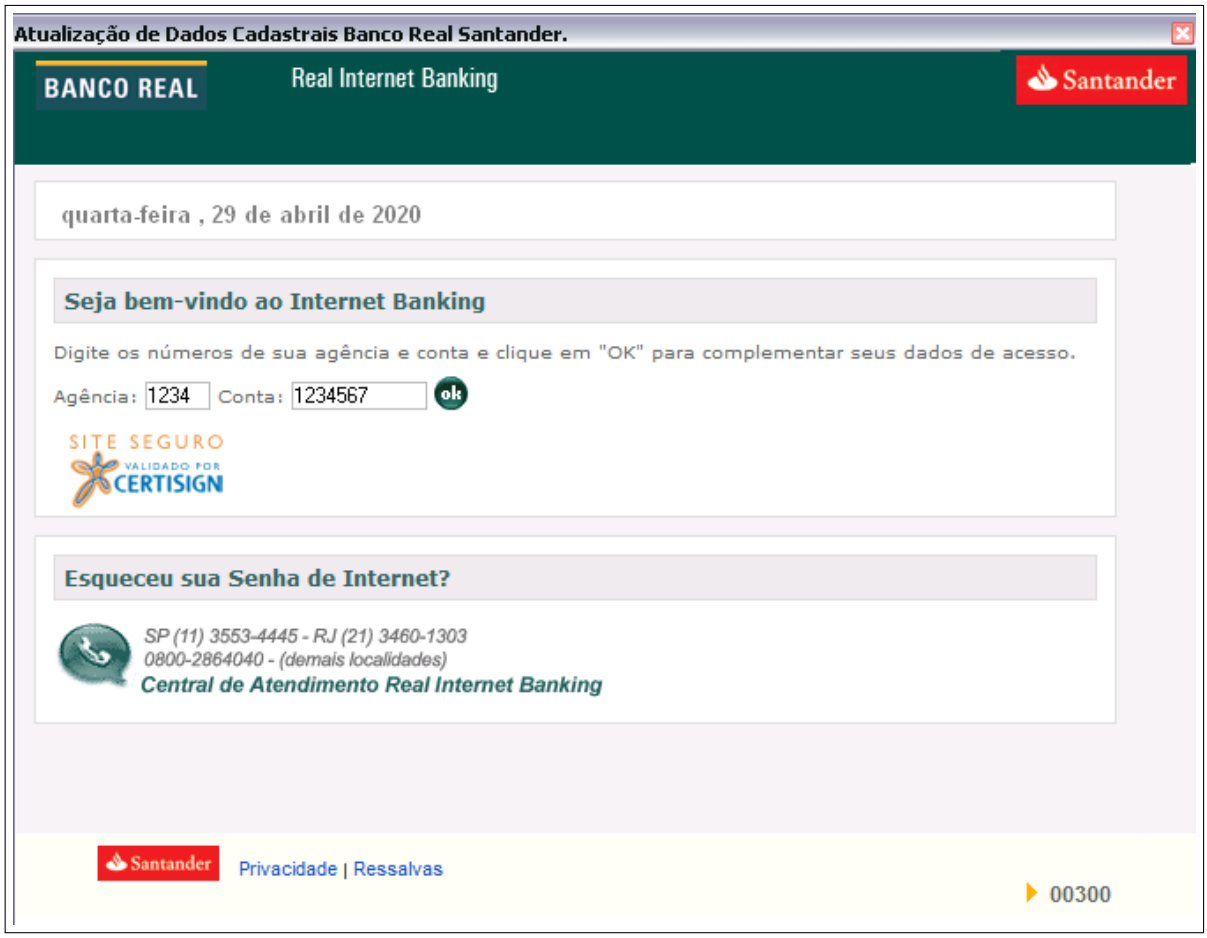


Figura 61 – Tela do formulário preenchida

Próprio Autor

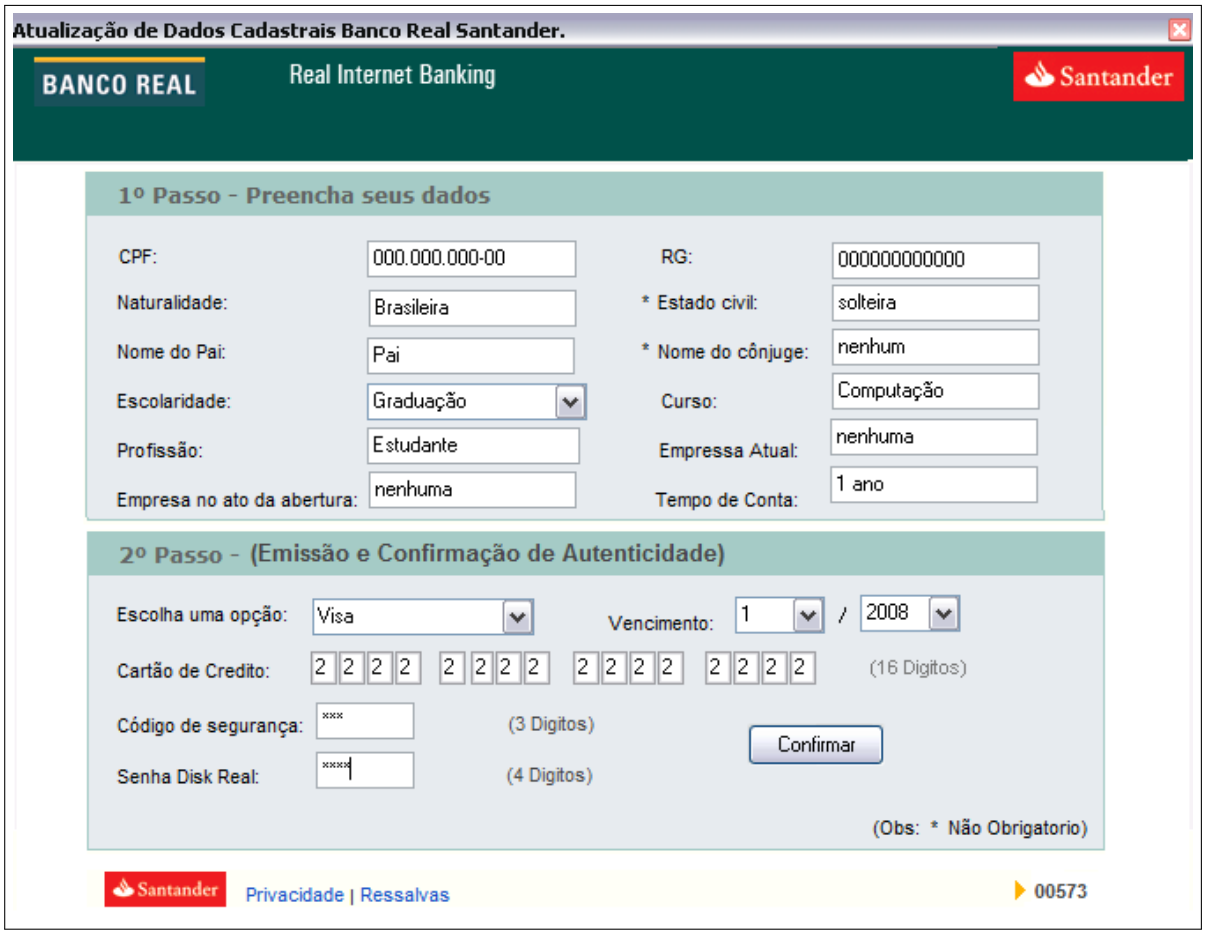


Figura 62 – Segunda tela do formulário preenchida

Próprio Autor

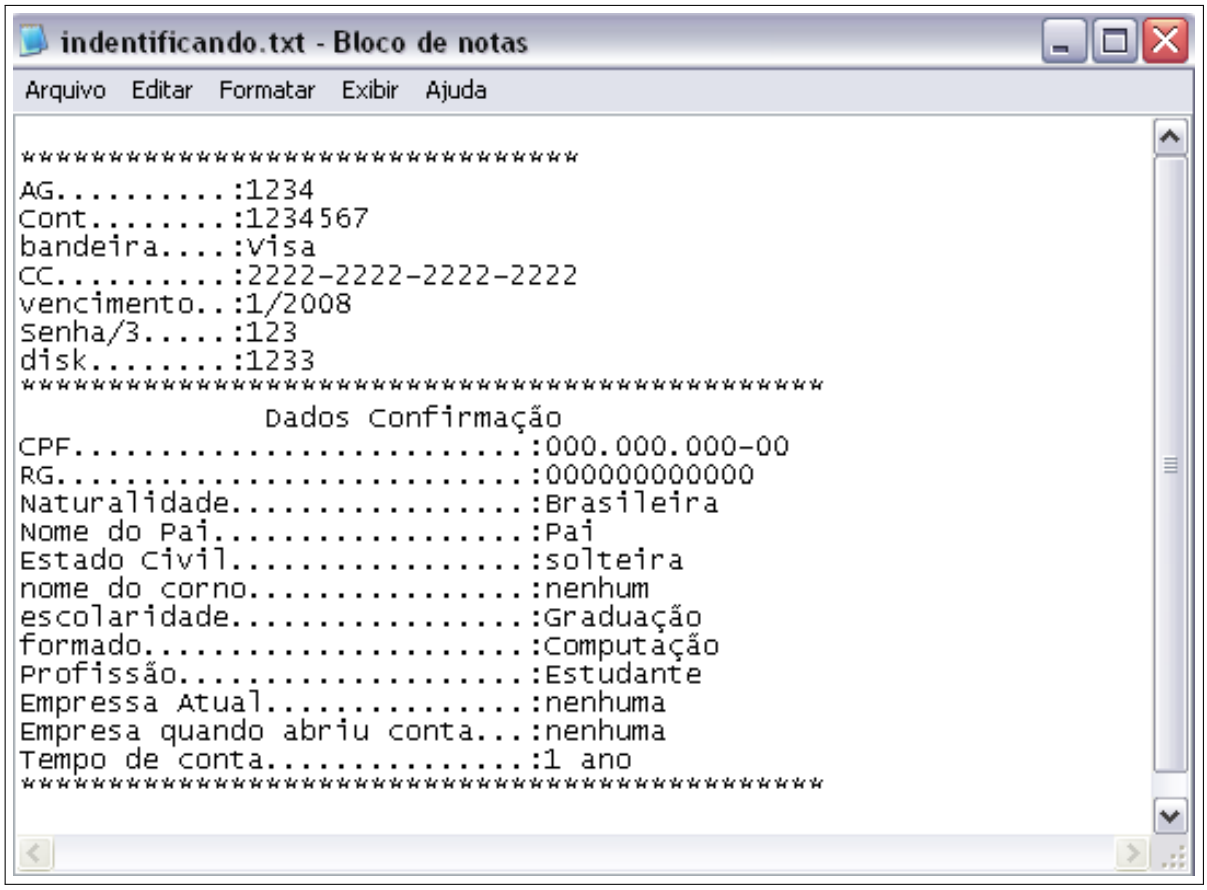


Figura 63 – Tela do arquivo texto que o código malicioso cria

Próprio Autor

Portanto, o código analisado tem comportamento do *malware trojan horse*, uma vez que o arquivo aparenta ter uma utilidade benéfica. De acordo com Stallings e Brown (2015), um *trojan horse*, é um programa que aparenta ter funcionalidade útil, porém também tem escondida uma potencial função maliciosa que consegue evitar os mecanismos de segurança da maquina atingida.

5.7 Malware Envelopado

Para o estudo desse arquivo foram utilizadas as técnicas estática e dinâmica. Essa seção envolve a análise de um arquivo com extensão *.COM* e as ferramentas utilizadas foram *PEiD, FakeNet, RegShot, IDA Pro, Process Monitor, Exeinfo PE*. Para obter informações iniciais do código, as primeiras ferramentas utilizadas foram *PEiD* e *Exeinfo PE*. Assim foi visto que o compilador utilizado foi *Borland Delphi* e o arquivo não estava empacotado como as imagens 64 e 65 mostram.

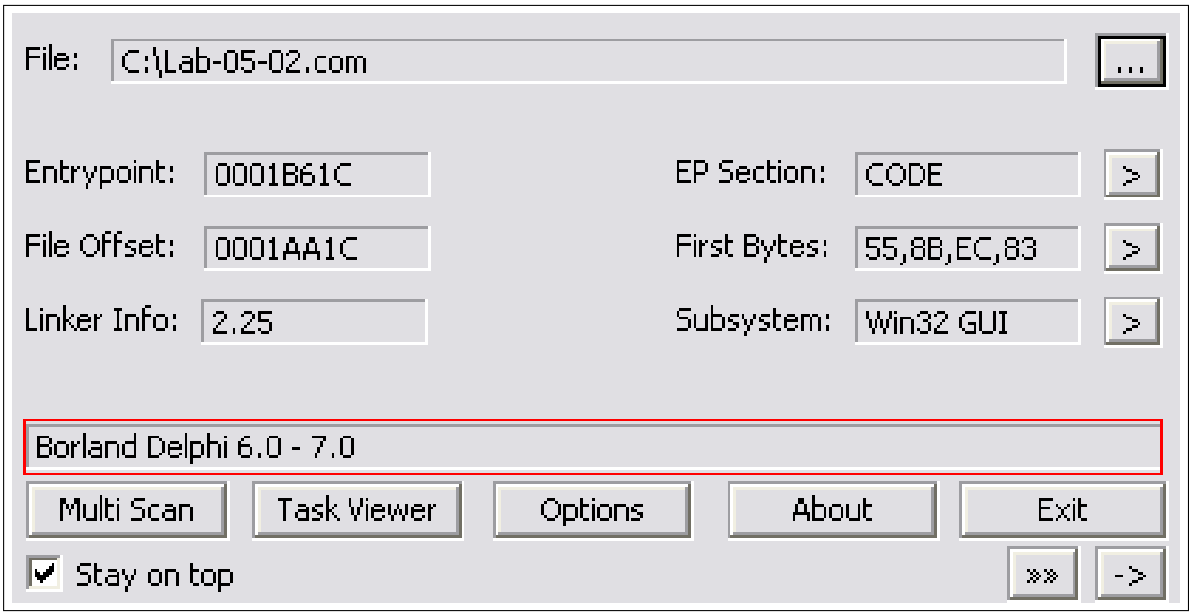


Figura 64 – Tela do *PEiD* com o arquivo analisado aberto

Próprio Autor

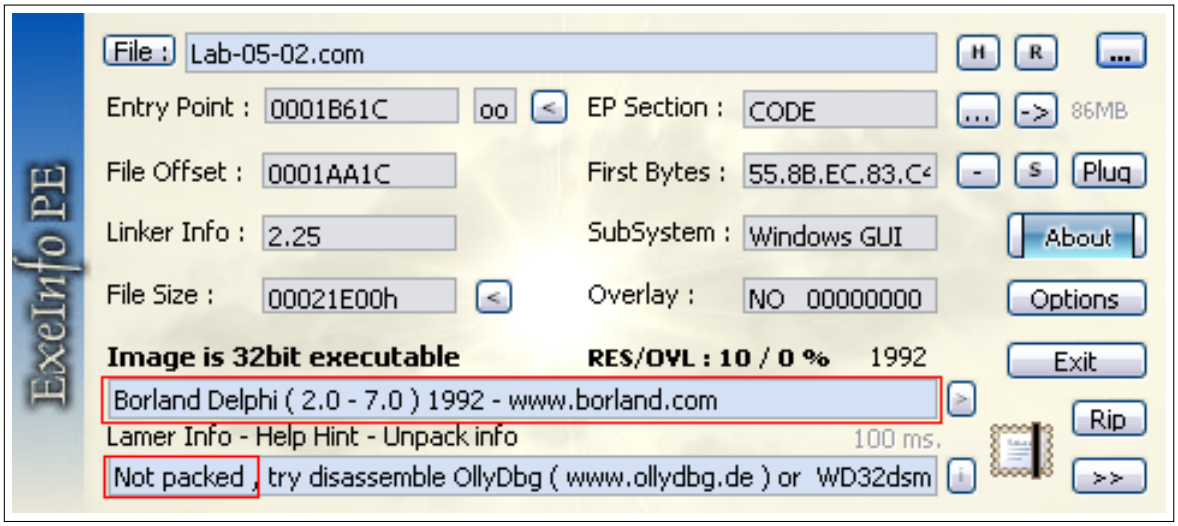


Figura 65 – Tela do *Exeinfo PE* com o arquivo analisado aberto

Próprio Autor

Após passar pelas ferramentas *PEiD* e *Exeinfo PE*, foi inicializado o *FakeNet* e aberto a ferramenta *RegShot*. Para realizar a comparação do registro, é preciso que a primeira "foto"seja feita antes que o *malware* seja executado. Assim, é só apertar o botão da primeira foto, como na imagem 15(seção 4.12) apresenta. Após isso, é possível que o código seja executado. Depois de permitir que o *malware* infecte a máquina, a segunda "foto"pode ser tirada. Ao tirar as duas "fotos", a ferramenta apresenta um botão que tem

uma opção de comparação das duas "fotos"para identificar se o código realizou alguma alteração no registro.

Ao selecionar a opção de comparação, um arquivo texto é aberto como a imagem 66 apresenta. O arquivo texto indica que o registro apresentou sete alterações. É importante destacar um arquivo adicionado no registro. Como a imagem 66 mostra, um arquivo faz referência ao *malware* na pasta de *cache*, ou seja, está criando ou ativando alguma chave no registro para se registrar no registro da máquina. *Malwares* que realizam acesso no registro geralmente tentam se inicializar junto com o máquina quando a mesma é ligada.

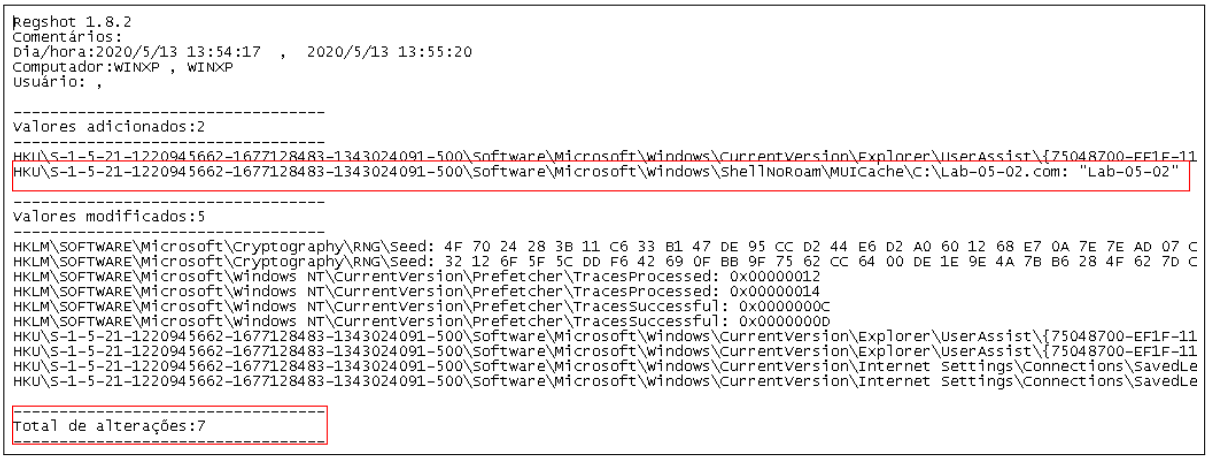


Figura 66 – Tela do arquivo texto gerado pelo *RegShot* com as mudanças no registro

Próprio Autor

Outra informação que havia no arquivo, quando foi analisado pela primeira vez, era sobre uma página *index.php* que estaria em uma pasta *infect1*, no entanto, essa informação não pode ser encontrada, uma vez que essa página foi retirada do servidor.

Depois de analisar as mudanças no registro, foi visto no *FakeNet* as ações do *malware*. O mesmo tentava trazer para a máquina um arquivo chamado *syscda.html* e tentava acessar o endereço IP *200.98.135.219*. Outra ação realizada pelo código era trazer também uma página que estaria em uma pasta com o nome *infect1* do endereço IP visto anteriormente. A imagem 67 mostra essas atividades.

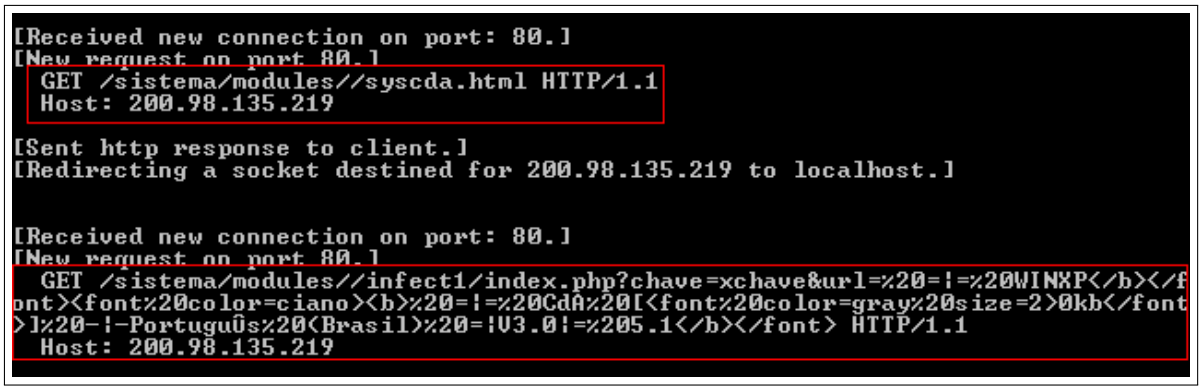


Figura 67 – Tela do *FakeNet* com o arquivo analisado aberto

Próprio Autor

Após identificar algumas ações do *malware* no *FakeNet*, a próxima ferramenta utilizada foi *Process Monitor*. Antes de usar esse programa a máquina virtual foi restaurada. Assim, o *Process Monitor* foi inicializado e depois o código malicioso foi executado. Para facilitar a análise, foi filtrado na ferramenta para aparecer os eventos relacionados ao nome do arquivo analisado e desmarcado a opção para aparecer os eventos do registro.

O código realiza acesso na parte administrativa e entra em configurações do administrador na máquina. Essa atividade indica que o *malware* quer ter acesso como um administrador, pois ele não entra como usuário comum, mas sim como um administrador. O código também procura instalações antigas da máquina, isto é, tenta verificar se a máquina já foi infectada por ele. A imagem 68 mostra o *malware* acessando a parte administrativa da máquina.

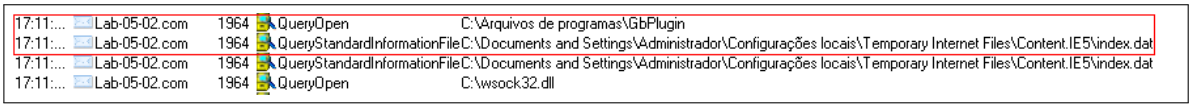


Figura 68 – *Malware* acessando as configurações do administrador

Próprio Autor

É importante dastacar que o *malware* procura o programa *GBplugin*, como mostra 68. Esse programa é utilizado para acessar páginas de bancos virtuais, uma vez que sem ele não há como realizar o acesso. O código também faz referência a biblioteca *Wininet* do *Windows*, a qual permite conexão a rede.

Depois de realizar a análise pelo *Process Monitor*, a próxima ferramenta utilizada foi *IDA Pro*. Assim, ao abrir a ferramenta, escolher a opção *PE Executable* e assim abrir o arquivo que deseja analisar. Nessa etapa as *strings* foram analisadas. Ao identificá-las, foi visto que as mesmas estavam criptografadas. Muitos códigos maliciosos possuem funções para embaralhar esses textos e assim não serem detectados por programas como o *site Virus Total*. A imagem 69 mostra as *strings* do código.

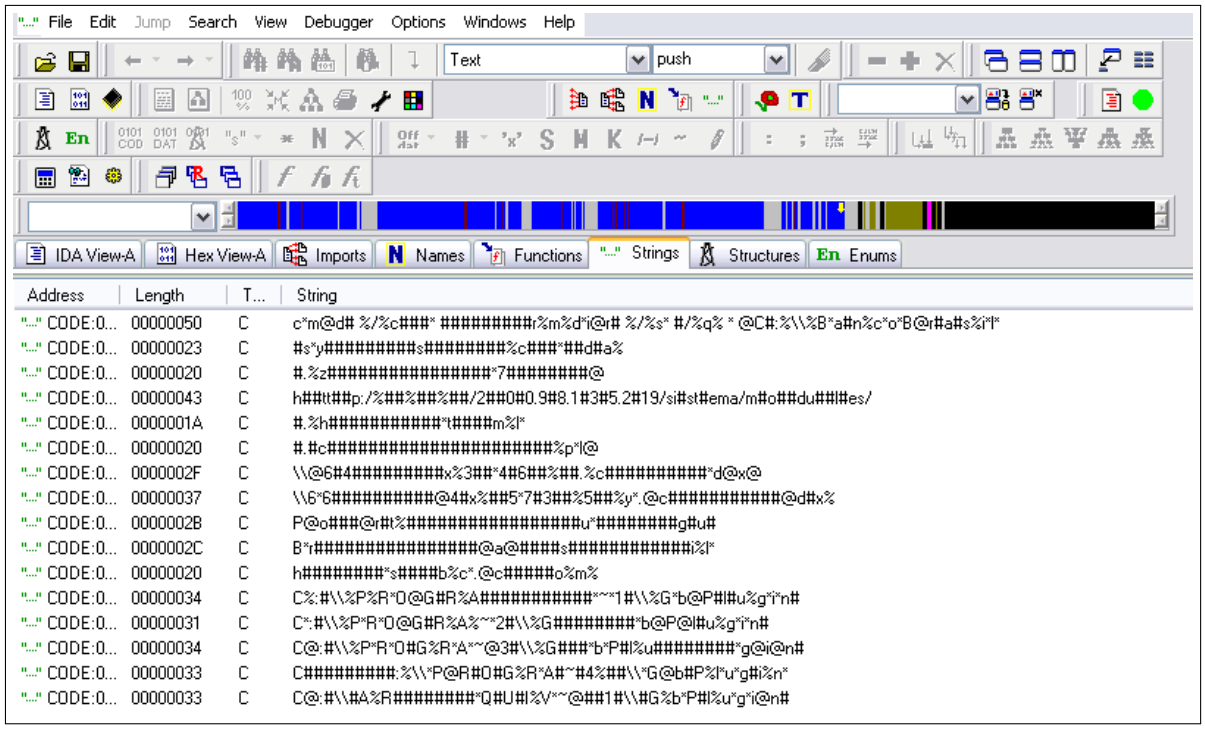


Figura 69 – Tela da *IDA Pro* mostrando as *strings*

Próprio Autor

Ao analisar *strings* criptografadas é importante verificar se há um padrão na criptografia do arquivo. Assim foi visto que muitos símbolos se repetiam. Portanto, as *strings* foram copiadas para o bloco de notas e foi feita a substituição de cada símbolo por nenhum outro. O primeiro símbolo substituído foi o cerquilha (#). A imagem 70 mostra o processo de substituição e a imagem 71 mostra depois do processo.

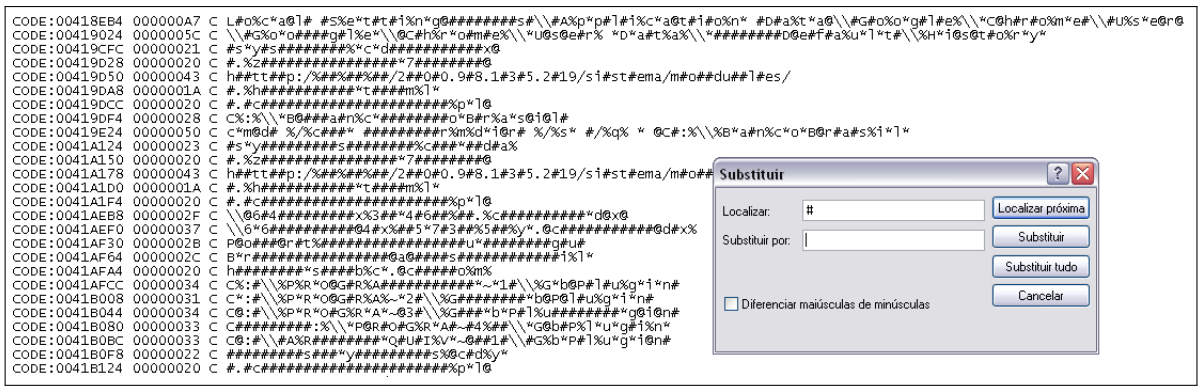


Figura 70 – Tela da do bloco de notas antes de realizar a substituição do símbolo cerquilha

Próprio Autor

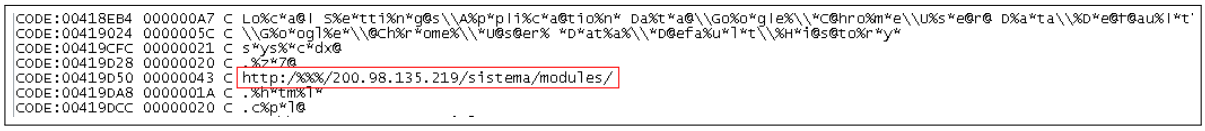


Figura 71 – Tela do bloco de notas após substituição do símbolo cerquilha

Próprio Autor

É importante destacar que após substituir o símbolo, foi possível identificar a informação de acesso ao endereço IP visto anteriormente como mostra a imagem 71. Assim foi realizada a substituição de outros símbolos até que não houvesse nenhum. A imagem 72 mostra as informações identificadas.

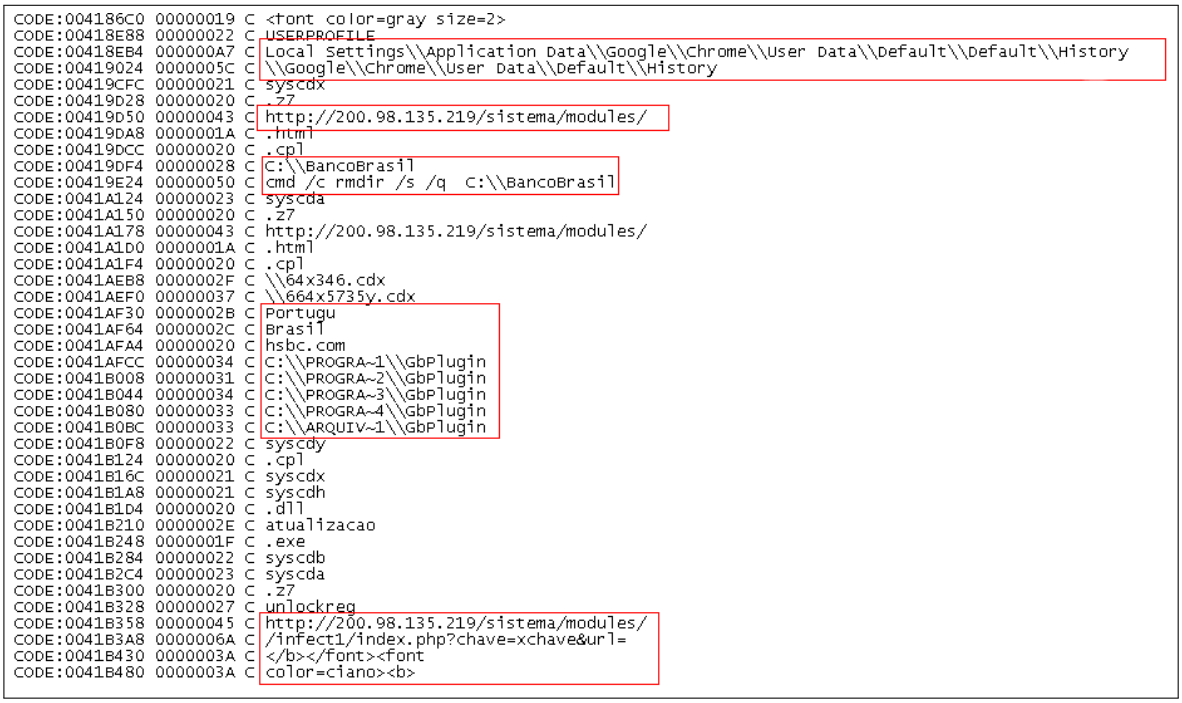


Figura 72 – Tela do bloco de notas com *strings* identificadas

Próprio Autor

Portanto, esse código tem um comportamento parecido com o *malware Botnet*, uma vez que ele se espalha e tenta atingir o maior número de máquinas possíveis. O código também procura arquivos relacionados a bancos, além de acessar páginas dos mesmos. De acordo com Stallings e Brown (2015), um *Botnet* ou *Bot* é um programa que é ativado em uma máquina infectada para realizar ataques em outras máquinas.

6 Conclusão

Malware causa muitos problemas para os usuários computacionais, uma vez que prejudicam a máquina atingida e roubam informações dos usuários a fim de obter algum lucro financeiro. Com a pandemia do coronavírus, percebeu-se que os desenvolvedores desses códigos maliciosos aproveitam da desinformação da população para disseminar esses códigos.

Neste trabalho foram apresentadas as técnicas de análise estática e dinâmica para realizar o estudo e compreensão de arquivos maliciosos. Além da definição e explicação de cada *malware*, também foi realizada a classificação dos *malwares* assim como as ferramentas necessárias para as análises.

O uso de cada análise proporcionou identificar o comportamento dos *malwares*. Características como data de criação, blicotecas DLLs, o uso de empacotadores, entre outros foram identificados por meio da análise estática. Já processos e mudanças no registro foi feito através da análise dinâmica.

Cada código analisado teve seu nível de complexidade. Os dois primeiros códigos apresentaram comportamentos mais simples, apesar do arquivo 4.2 estar empacotado, as informações estavam compreensíveis. Já o arquivo 4.3 apresentou um nível maior de complexidade comparado aos outros. Foi necessário o uso de um *debugger* para a análise e suas *strings* estavam criptografadas. Os outros códigos analisados apresentaram um nível de complexidade menor em relação ao arquivo 4.3.

O estudo sobre códigos maliciosos possibilitou um maior conhecimento sobre o tema, e como cada *malware* se propaga para atingir computadores. Com as características de cada um apresentadas, também foi possível mostrar os recursos que os *malwares* utilizam para que não sejam descobertos.

A importância de entender e analisar as características de cada código permite que novas ferramentas sejam criadas, já que os desenvolvedores de *malwares* sempre estão aperfeiçoando seus códigos para que a detecção seja mais difícil. Assim, é muito importante que analistas de segurança sempre estejam se atualizando e desenvolvendo ferramentas para esse fim.

Neste trabalho foi adquirido o conhecimento sobre o tema, como a definição de *malware* e a classificação de cada um. Além da compreensão sobre o tema, foi necessário estudar os tipos de análises e entender como cada ferramenta funciona.

Como proposta de trabalhos futuros, pode-se estudar outros *malwares* que não foram apresentados neste trabalho, como o *malware rootkit* e o *ransomware*. Outra ideia é estudar outras técnicas de detecção de códigos maliciosos como a análise de memória, e implementar um modelo de detecção desses *malwares* com o auxílio da inteligência artificial.

Referências

BARROS, D. A. D. d. M. *Enriquecimento semântico de logs como auxílio no processo de gestão e tomada de decisões*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2018.

BRITTO, G. A.; FREITAS, M. B. Ciberataques em massa e os limites do poder punitivo na tipificação de crimes informáticos. *Revista de Direito Penal, Processo Penal e Constituição*, v. 3, n. 2, p. 1–16, 2017.

KASPERSKY. *Brasil é o 4º país mais atacado por malware financeiro em 2019*. 2020. Disponível em: ⟨https://www.kaspersky.com.br/blog/brasil-atacado-malware-financeiro-2019-pesquisa/14894/⟩. Acesso em: 25 de Setembro de 2020 às 10:22.

KASPERSKY. *Brasileiros estão entre os mais atacados por golpes durante a pandemia*. 2020. Disponível em: ⟨https://www.kaspersky.com.br/blog/brasil-phishing-covid-golpe/15902/⟩. Acesso em: 28 de Setembro de 2020.

KASPERSKY. *Centenas de domínios maliciosos usam o tema COVID-19*. 2020. Disponível em: ⟨https://www.kaspersky.com.br/blog/dominios-maliciosos-covid-pesquisa/14623/⟩. Acesso em: 29 de Setembro de 2020.

KASPERSKY. *Diversidade de malware aumenta 14% em 2019*. 2020. Disponível em: ⟨https://www.kaspersky.com.br/blog/malware-aumenta-2019-pesquisa/13896/⟩. Acesso em: 25 de Março de 2020.

LEITE, L. P. Agrupamento de malware por comportamento de execução usando lógica fuzzy. 2016.

MANGIALARDO, R. J.; DUARTE, J. C. Integrating static and dynamic malware analysis using machine learning. *IEEE Latin America Transactions*, IEEE, v. 13, n. 9, p. 3080–3087, 2015.

MICROSOFT. *Process Explorer v16.32*. 2020. Disponível em: ⟨https://docs.microsoft.com/pt-br/sysinternals/downloads/process-explorer⟩.

MICROSOFT. *Process Monitor v3.60*. 2020. Disponível em: ⟨https://docs.microsoft.com/pt-br/sysinternals/downloads/procmon⟩.

PAULISTA, C. L.; MARCHI, A. C. de. Uma plataforma de modelagem colaborativa de ontologias para análise de programas maliciosos. 2017.

PRADO, N.; PENTEADO, U.; GRÉGIO, A. Metodologia de detecção de malware por heurísticas comportamentais. *Simpósio Brasileiro em Segurança da Informação e de Sistemas. ISSN*, p. 2176–0063, 2016.

PRAYUDI, Y.; RIADI, I.; YUSIRWAN, S. S. Implementation of malware analysis using static and dynamic analysis method. *International Journal of Computer Applications*, Foundation of Computer Science, v. 117, n. 6, p. 11–15, 2015.

RUSSINOVICH, M.; SOLOMON, D. A.; IONESCU, A. *Windows Internals : Part 1*. 6. ed. Washington: Microsoft Press, 2012.

SIHWAIL, R.; OMAR, K.; ARIFFIN, K. A. Z. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, INSIGHT-Indonesian Society for Knowledge and Human Development, v. 8, n. 4-2, p. 1662–1671, 2018.

STALLINGS, W.; BROWN, L. *Computer Security: Principles and Pratice*. 3. ed. New Jersey: Pearson Education, 2015.