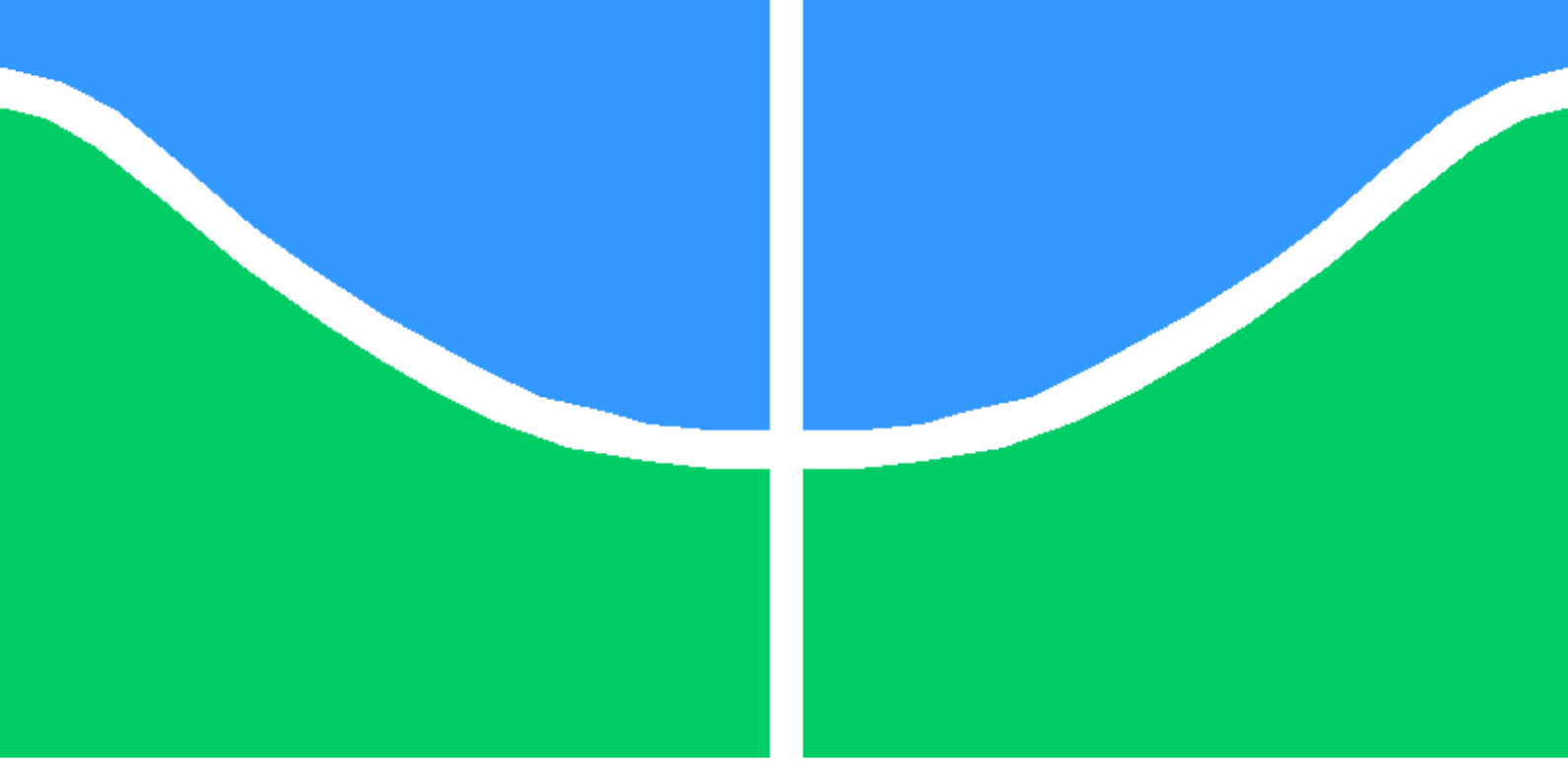


TRABALHO DE GRADUAÇÃO

TÉCNICAS E FERRAMENTAS DE ANÁLISE VISUAL DE *MALWARES*

Victor Hugo de Souza

Brasília, Dezembro de 2016

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

TÉCNICAS E FERRAMENTAS DE ANÁLISE VISUAL DE *MALWARES*

Victor Hugo de Souza

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Flavio Elias de Deus, ENE/UnB _____
Orientador

Prof. Robson Albuquerque, ENE/UnB _____
Co-Orientador

Prof. Fábio Mendonça, ENE/UnB _____
Examinador interno

Dedicatória

Dedico esse trabalho a todos os leitores que buscam mais conhecimento na área de Redes de Comunicação e a todos que acreditam na importância da educação para o desenvolvimento do nosso país.

Victor Hugo de Souza

Agradecimentos

Agradeço primeiramente a minha família, por terem me dado tanto apoio para chegar até aqui, principalmente aos meus pais, Adilson e Andreia, que sempre me proporcionaram as melhores oportunidades de estudo e que me mostram como ser uma pessoa melhor todos os dias. Agradeço os professores Flavio Elias e Robson Albuquerque por compartilharem seus conhecimentos, serem flexíveis e me guiarem para que pudesse concluir este trabalho. Agradeço também o Laboratório Forense de Dispositivos Computacionais pelo apoio fornecido para a realização deste trabalho. Agradeço à Camila Nakano, que esteve sempre presente na realização deste trabalho dando apoio, mesmo nos momentos difíceis. Agradeço ao Henrique Primo, meu ex-supervisor de estágio, por apoiar durante quase toda a graduação e fornecer uma oportunidade profissional. Por fim agradeço aos colegas que conheci ao longo do curso e que caminharam junto a mim nessa jornada e a todos os amigos que de alguma forma contribuíram para que eu pudesse concluir mais essa etapa da minha vida me proporcionando momentos de muita alegria.

Victor Hugo de Souza

RESUMO

Devido ao crescimento de ameaças de códigos e programas maliciosos, o monitoramento de redes, computadores e dispositivos móveis está se tornando cada vez mais importante. Pesquisadores podem precisar utilizar uma análise manual estática ou dinâmica para estudar novas amostras. Quanto maior o for o tempo utilizado para analisar um *malware*, maior será o dano causado. Este crescimento de ameaças fez com que surgissem ferramentas cujo propósito é analisar tais *malwares*. Estas ferramentas têm como objetivo auxiliar os analistas para rapidamente detectarem e classificar uma nova amostra de *malware*. A visualização dos dados e o uso de métodos de análise visual durante a exploração de dados são abordagens que podem auxiliar a análise de novos programas maliciosos. Existem diferentes métodos de visualização disponíveis que fornecem interação para a exploração de dados. Este trabalho apresentará uma visão geral das técnicas de visualização e ferramentas existentes para análise de *malware* a partir da análise visual. Após obter os resultados experimentais será verificada a capacidade das ferramentas em destacar os pontos relevantes nas imagens e a aceitabilidade de personalização das imagens criadas. Com base em todos os resultados esperasse comprovar a eficácia dos métodos de análise visual para auxiliar e complementar as técnicas atuais.

Palavras-chave: Redes de Comunicações, *Malwares*, Análise Visual.

ABSTRACT

Due to the growth of threats and malicious programs, monitoring of networks, computers and mobile devices is becoming increasingly important. Researchers may need to use a static or dynamic manual analysis to study new samples. The longer the time consumed to analyze malware, the greater the damage caused. This growth of threats made to appear tools whose purpose is to analyze such malware. These tools are intended to assist the analysts to quickly detect and classify a new malware sample. The visualization of data and the use of visual methods of analysis for the exploration of data are approaches that can assist the analysis of new malicious programs. There are different preview methods available that provide interaction for data exploration. This paper will present an overview of visualization techniques and existing tools for malware analysis from the view of visual analysis. After obtaining the experimental results will be verified the ability of tools to highlight relevant points in the images and the acceptability of customization of images created. Based on all the results expected prove the effectiveness of the methods of visual analysis to assist and complement the current techniques.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

Acrônimos

API	Application Programming Interface
DLL	Dynamic Link Library
ELF	Executable and Linking Format
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
MD5	Message-Digest algorithm 5
PE	Portable Executable
SHA	Secure Hash Algorithm
TLS	Thread-Local Storage
UPX	Ultimate Packer for Executables
USL	UNIX System Laboratories

Capítulo 1

Introdução

Atualmente há um crescente uso de dispositivos conectados à internet. Novos dados de uma pesquisa da Juniper revelou que o número de dispositivos IoT (do inglês, *Internet of Things*) irá alcançar a marca de 38,5 bilhões em 2020 [1]. Este crescimento aumenta o número de objetos expostos a infecção de *malwares*, conforme é possível observar no estudo realizado pela Nokia, o número de dispositivos infectados aumentou 96% na primeira metade de 2016 comparado com a segunda metade de 2015, de 0,25% para 0,49% [2].

Os *malwares* estão se tornando mais sofisticados, utilizando diversos métodos para infringir os bloqueios. Além disso, a diversidade de *malwares* também tem crescido junto com a evolução dos mesmos. Comparando o total de *malwares* entre os anos de 2012 e 2016, ocorreu um crescimento de aproximadamente 580%, conforme pode ser observado na Figura 1.1. Assim, é possível concluir que a complexidade dos *malwares* tem aumentado proporcionalmente a sua variedade, os tornando cada vez mais difíceis de serem combatidos[3].

Com tantos *malwares* capazes de acessar e explorar sistemas vulneráveis, é preciso mais do que programas que possam prevenir e defender os dispositivos, sendo necessário que *malwares* possam ser reconhecidos e analisados para posteriormente serem mapeados a fim de sejam descobertas maneiras de evitá-los. O processo de detecção e criação de uma assinatura para um novo *malware* é demorado e pode exigir uma análise manual das amostras, entretanto, quanto mais demorado o processo de detecção e análise, maior o tempo para que os desenvolvedores desses *malwares* possam aprimorá-los, ficando cada vez mais complicado extingui-los.

Há duas abordagens para análise de uma amostra de *malware*. Estas são a análise dinâmica e a análise estática. A análise dinâmica é uma técnica para estudar o comportamento de uma amostra de *malware* enquanto a amostra está sendo executada. A análise estática, por outro lado, é uma técnica que permite o estudo de uma amostra, sem a necessidade de execução da amostra. Muitas vezes, não é possível evitar a análise manual, especialmente em novos *malwares*. As duas abordagens, quando feitas manualmente vão exigir muito tempo.

Com o crescente aumento do número de amostras de *malwares* para processar, é necessária uma nova técnica para ajudar na análise. Uma solução possível para auxiliar a análise de amostras de *malwares* é através do uso de visualização de *malwares*. Visualização de *malwares* é um campo

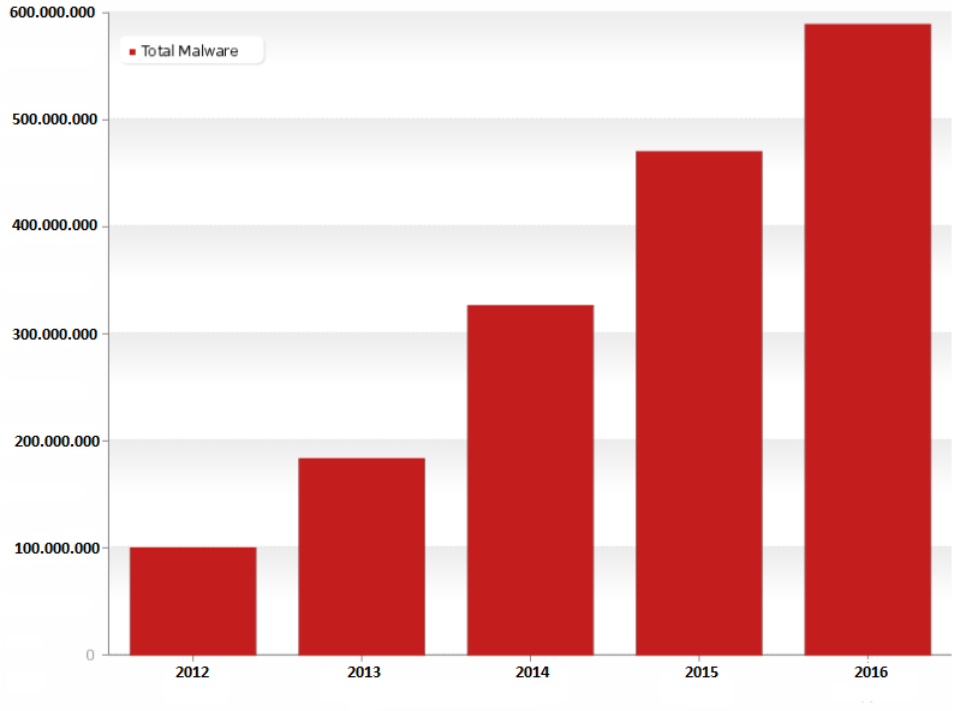


Figura 1.1: Estimativa do total de *malwares* nos últimos 5 anos[3].

de conhecimento focado em representar características de *malwares* na forma de pistas visuais, que poderiam ser usadas para fornecer mais informações sobre um *malware* específico em uma forma mais compacta. A visualização ajuda os pesquisadores a entender melhor o *malware* graficamente, destacando os aspectos mais marcantes de um *malware* que podem não ser concebíveis em outras formas de análise. Isto pode levar a mais conhecimento a ser extraído a partir da mesma quantidade de dados, e assim, contribuir para uma melhor e mais ágil compreensão do funcionamento de um programa malicioso.

1.1 Objetivos

Este trabalho tem como objetivo apresentar e analisar os resultados obtidos ao realizar análise visual de *malwares* utilizando diferentes técnicas e ferramentas, com o intuito de validar a eficácia deste método de análise. Os principais objetivos desse trabalho são:

- Verificar a capacidade das ferramentas em destacar os pontos relevantes nas imagens;
- Verificar a capacidade das ferramentas em aceitar personalização;
- Verificar a flexibilidade das ferramentas em aceitar como entrada diversos tipos de arquivos através da configuração adequada de parâmetros;
- Analisar diferentes famílias de *malwares*, com ênfase nos métodos utilizados por cada ferramenta;

1.2 Estrutura do Trabalho

No Capítulo 2, Fundamentação Teórica, são abordados os principais fundamentos estudados e utilizados nesse projeto.

No Capítulo 3, Métodos Propostos, são apresentadas as amostras *malware* e as ferramentas utilizadas, bem como um roteiro de uso das ferramentas.

No Capítulo 4, Resultado Experimentais, será feita uma comparação entre as diferentes técnicas e classes de *malwares* com base nas imagens obtidas.

Por último, o Capítulo 5, Conclusão, contém as principais conclusões obtidas com as simulações e os resultados analisados. Esse capítulo também contém as perspectivas de trabalhos futuros a partir desse projeto.

Capítulo 2

Fundamentação Teórica

Esse capítulo apresenta os principais fundamentos teóricos nos quais esse trabalho foi embasado. Apresentamos nesse capítulo os conceitos e mecanismos de funcionamento dos recursos utilizados ao longo do desenvolvimento desse trabalho. Assim, os conceitos mais estudados e necessários para o desenvolvimento do projeto são expostos de forma a trazer um conhecimento básico teórico para o entendimento das análises realizadas.

2.1 Técnicas de Visualização

Existem diversas técnicas de visualização de dados, desde as mais simples e genéricas, como áreas, barras, linhas e pontos, até as mais complexas com representações em 3D. Técnicas de visualização podem ser separadas em categorias, porém algumas técnicas pertencem a mais de uma categoria, e outras a nenhuma delas. Como existem diversas técnicas, será abordado apenas um conjunto delas.

2.1.1 Técnicas Geométricas

Através de transformações geométricas dos valores de seus atributos, permitem a visualização dos dados. Matriz de espalhamento é uma das técnicas mais conhecidas desta categoria, que apresenta uma matriz de plotagens bidimensionais de dados, onde cada combinação de duas dimensões é representada por uma plotagem. Nesta técnica é possível observar correlações diretas e inversas entre atributos, distribuição de valores nos atributos, padrões e exceções e outros fenômenos que podem servir para sugerir hipóteses sobre os dados.

Outra técnica geométrica bastante utilizada é a de coordenadas paralelas. Nela é criado um eixo para cada um dos atributos, estes eixos são organizados de forma paralela e equidistante, e para cada dado a ser visualizado é traçada uma linha poligonal que cruza os eixos na altura dos valores correspondentes para aqueles atributos. Neste tipo de técnica é possível visualizar a correlação entre atributos, padrões e exceções. Também é possível inferir métricas para separação de tráfego suspeito, pois a diferença do tipo de tráfego pode ser claramente observada nos eixos de

alguns dos atributos. A figura 2.1 mostra a utilização desta técnica para analisar uma amostra de um *adultbrowser*.

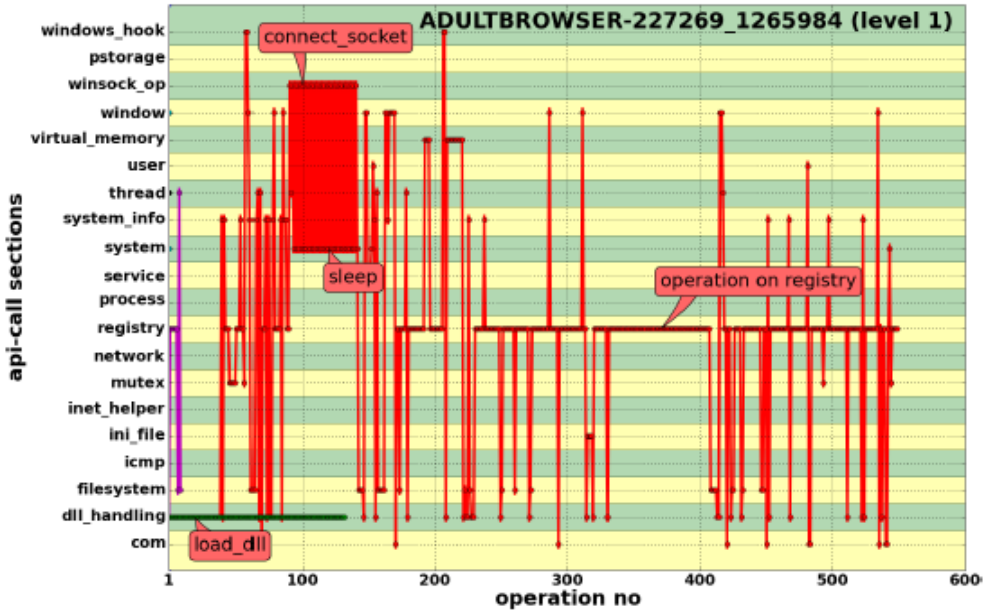


Figura 2.1: Representação gráfica de um *adultbrowser* usando *thread graph*[4].

Ao processar uma enorme quantidade de dados, estas técnicas irão inserir muitas linhas no desenho, o que irá dificultar na análise do usuário. Outra dificuldade desta técnica, é a ordem em que as linhas são desenhadas, dependendo da ordem podem causar sobreposiçãoentre elas, não só dificultando a visualização, mas impedindo em alguns casos.

2.1.2 Técnicas Baseadas em Ícones

Os dados multidimensionais são representados como ícones cujas características correspondem aos valores dos atributos dos dados. Chernoff faces[5] é a técnica mais conhecida desta categoria, esta técnica utiliza pequenos ícones de faces para representar os valores dos dados. Um exemplo desta técnica é apresentado na Figura 2.2. Codificação por formas é outra técnica baseada em ícones, ela mapeia os valores multidimensionais em um pequeno gráfico retangular que é usado como marcador em um gráfico bidimensional, cujas dimensões podem ser espaciais ou temporais.

O grande desafio destas técnicas é a representação adequada para as características dos ícones, que possibilite uma interpretação visual fácil através de comparação com outros ícones para que possa identificar similaridades e exceções. Estas técnicas exigem um constante uso de uma legenda.

2.1.3 Técnicas Baseadas em *Pixel*

Técnicas bastante similares às que usam ícones, pois para representar os valores multidimensionais utilizam pequenos conjuntos em um arranjo espacial e organizam estes conjuntos em arranjos

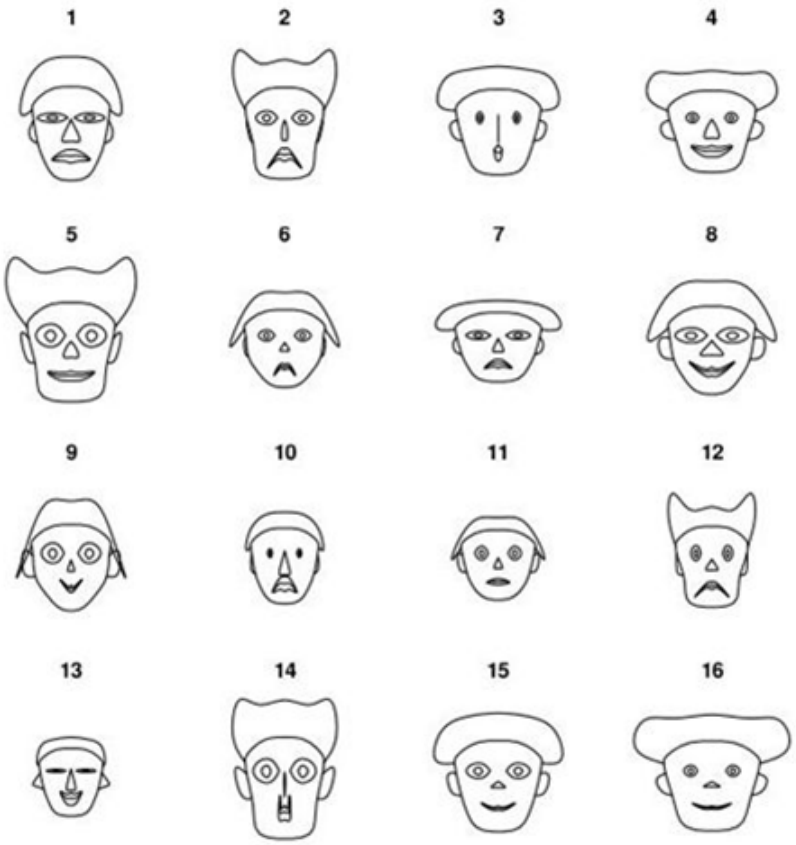


Figura 2.2: Exemplo da técnica Chernoff faces[5].

maiores que podem representar as dimensões, sejam elas espaciais ou temporais, do conjunto de dados. Como os valores dos atributos são representados por *pixels* coloridos, é necessário um mapa de cores para facilitar a identificação de valores similares e diferentes. A analise é feita usando atributos visuais como continuidade e textura para identificar padrões. Nas Figuras 2.3 e 2.4, é possível observar o resultado obtido ao processar os executáveis dos programas: calculadora.exe, cmd.exe, notepad.exe e chrome.exe.

2.1.4 Técnicas Hierárquicas

Particiona as múltiplas dimensões dos dados de forma hierárquica em subespaços que podem ser visualizados. *Treemap*[4] é uma das técnicas hierárquicas mais conhecidas, este método exibe dados como um conjunto de retângulos aninhados, para cada ramo da árvore é definido um retângulo que é então preenchido com retângulos menores lado a lado, os quais representam sub-ramos. Outras características como cores ou texturas podem ser usadas para denotar informações adicionais na visualização. Na figura 2.5 é possível observar duas seções que são significantemente mais largas que as outras, indicando que aquela amostra realiza algumas operações específicas com bastante frequência.

O dendograma[6] é outra técnica hierárquica bem conhecida, estes gráficos particionam hierar-

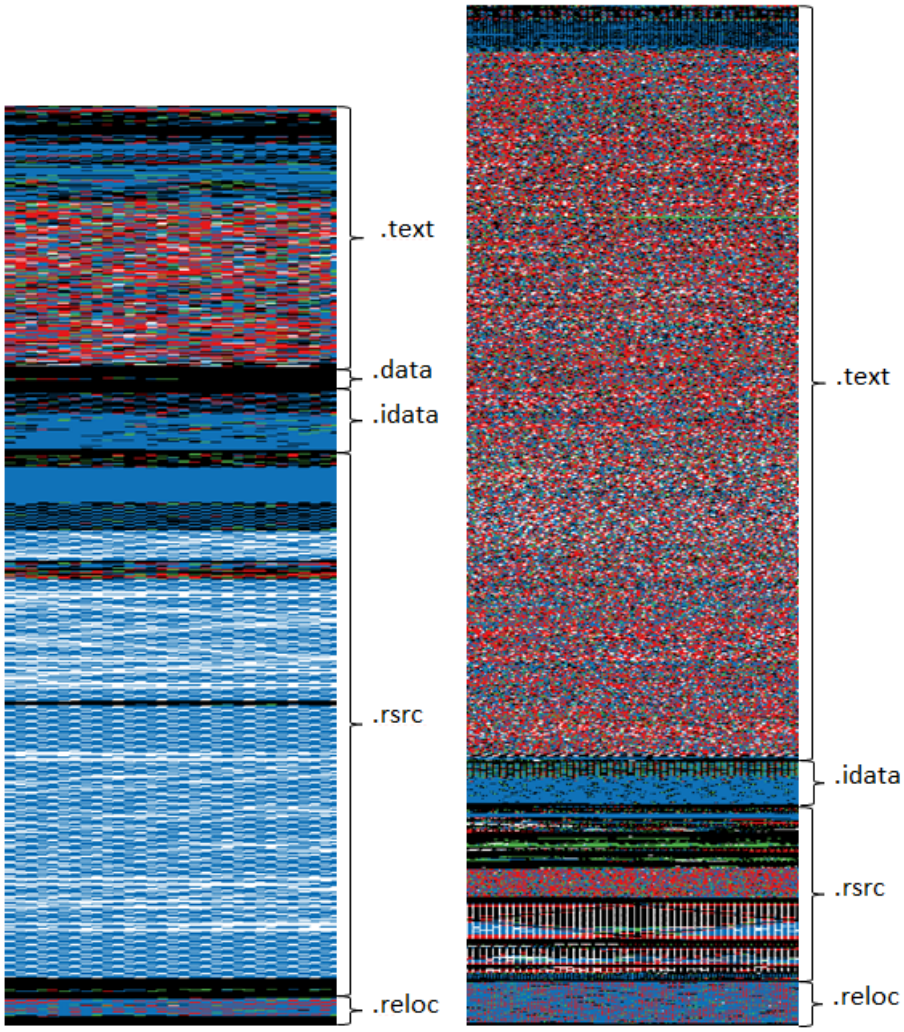


Figura 2.3: Análise Calculadora (à esquerda) e CMD (à direita) utilizando a ferramenta binvis.io.

quicamente um conjunto de dados, mostrando agrupamentos e distâncias entre os dados.

2.1.5 Técnicas Baseadas em Grafos

Nestas técnicas, a representação é feita através de vértices (objetos) e arestas (relação), permitindo a visualização de padrões e valores associados às relações. Um exemplo de visualização de similaridade entre *malwares*, utilizando a API JUNG[8], pode ser visto na Figura 2.7. A maior dificuldade desta técnica é a visualização, porém existem estudos de algoritmos para posicionamento de vértices e arestas de forma a facilitar a visualização do conjunto.

2.1.6 Técnicas Tridimensionais

Utiliza gráficos tridimensionais para visualização, onde os dados são organizados em tipos de cenários. A grande vantagem destas técnicas é serem interativas permitindo o usuário selecionar

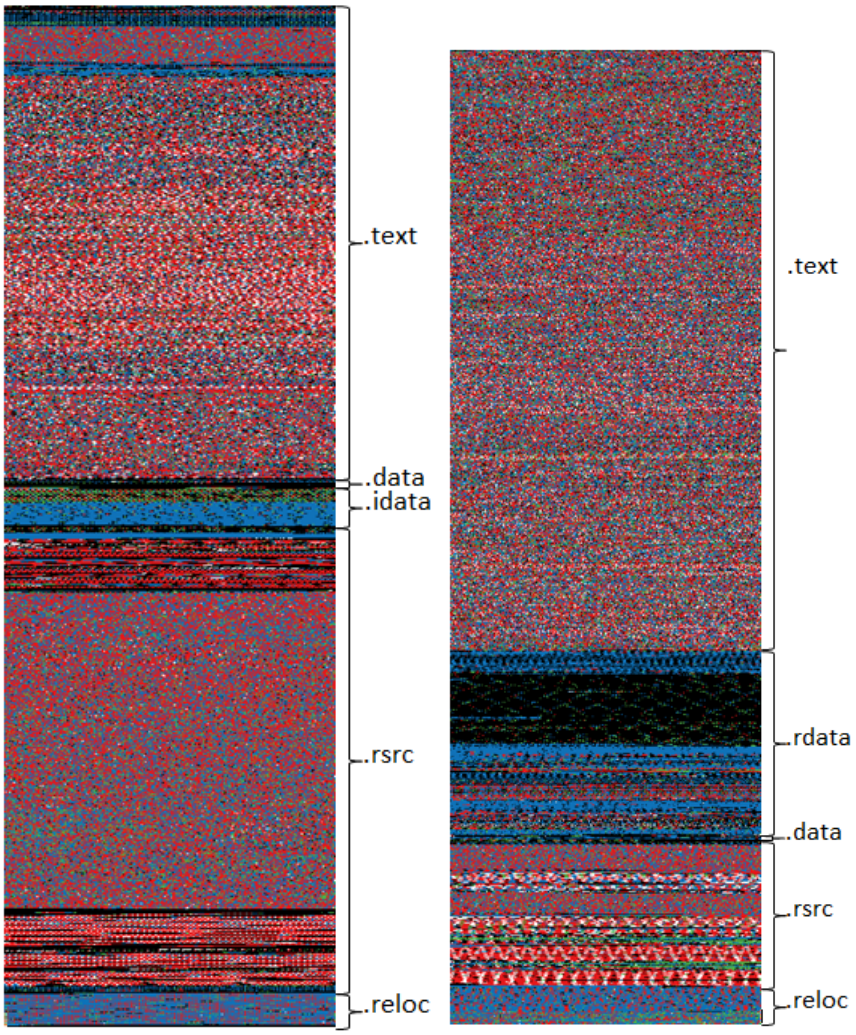


Figura 2.4: Análise Notepad (à esquerda) e Chrome (à direita) utilizando a ferramenta binvis.io.

uma região ou subconjunto de dados que irão proporcionar a melhor visualização. Para um grande volume de dados, estas técnicas exigem um hardware específico ou um computador com uma quantidade razoável de processamento.

2.2 *Malware*

Malware é uma abreviação para "*software* malicioso". É um *software* destinado a infectar ou realizar ações não desejadas a um sistema computacional. Exemplos comuns destes *softwares* maliciosos são: vírus, *worms*, *trojan* e *spyware*.

É lamentável que existam programadores com intenção maliciosa, mas é bom estar ciente do fato. Hoje existem vários utilitários como antivírus e *antispyware*, que permitem o sistema procurar e destruir os programas maliciosos que são encontrados. Os *malwares* serão o foco deste trabalho, onde serão abordadas técnicas para auxiliar na descoberta de novas ameaças de forma mais eficaz.

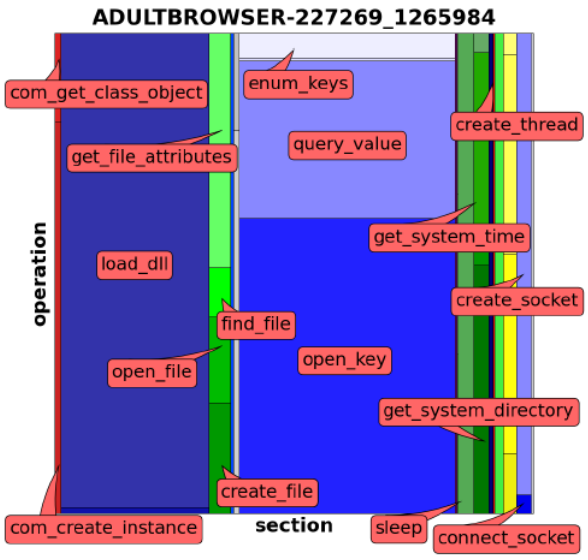


Figura 2.5: Representação gráfica de um *adultbrowser* usando *Treemap*[4].

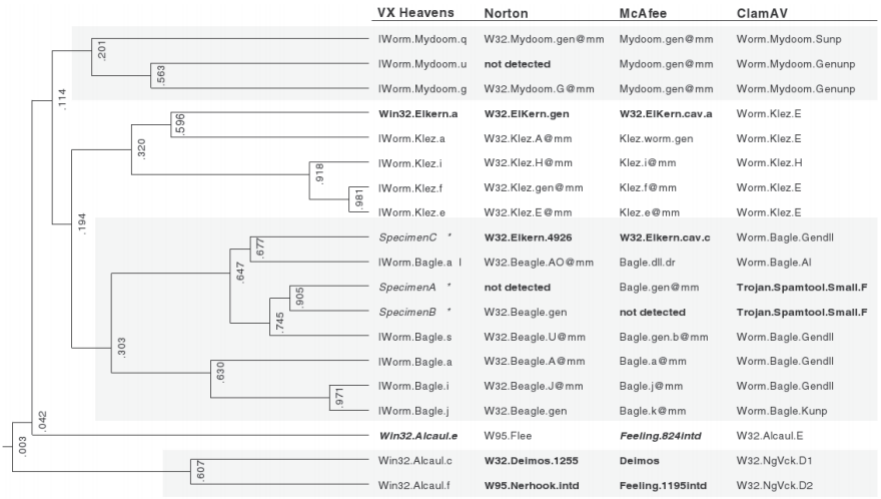


Figura 2.6: Dendograma mostrando um modelo de filogenia de *malware* e respectiva classificação por alguns anti-virús[7].

2.2.1 Tipos de *Malwares*

Segundo Michal Sikorski[10] os *malwares* podem ser classificados com base seu comportamento, nas próximas seções será feito um breve detalhamento sobre as categorias mais conhecidas.

É importante não fixar em uma classificação, pois os *malwares* podem ser classificados de diversas formas, por exemplo um programa pode ter um *keylogger* para coletar senhas e um *worm* para enviar spans.

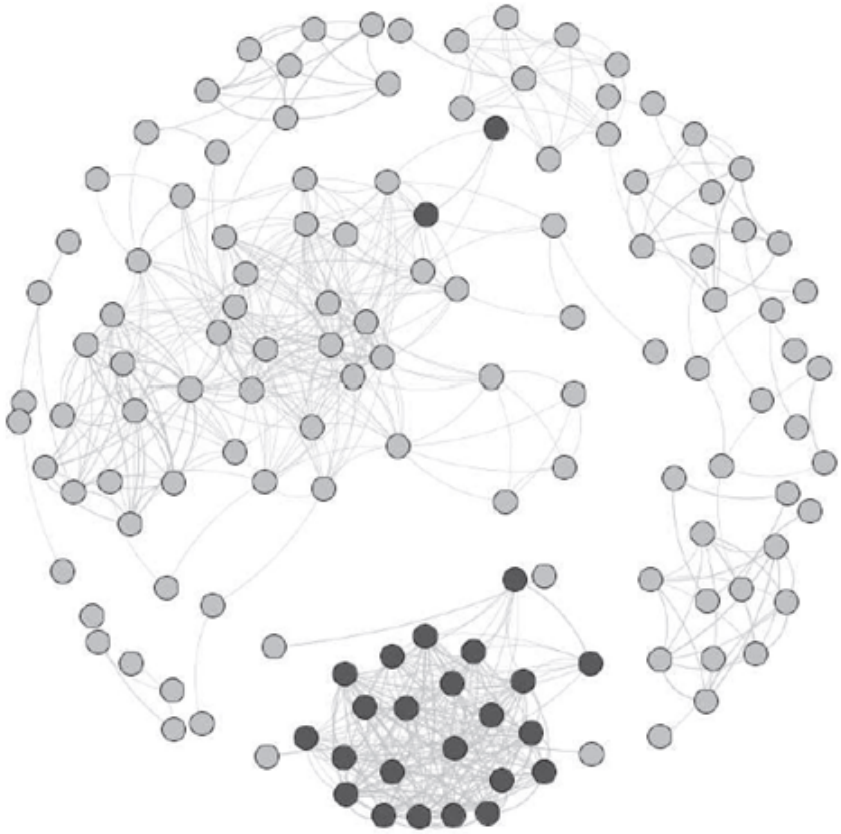


Figura 2.7: Visualização de similaridade entre *malwares* com a API JUNG[9].

2.2.1.1 Vírus

Vírus é um código malicioso auto replicante que pode ser identificado de acordo com o método de infecção e seleção do objeto a ser infectado, e se propaga fazendo cópias de si mesmo e se tornando parte de outros programas. Na maioria das vezes, eles se utilizam de falhas de segurança existentes em programas ou sistemas operacionais. A maioria dos vírus estão vinculados a um arquivo executável, o que significa que um vírus pode existir em um sistema, porém só terá a capacidade de infectá-lo se o usuário executar ou abrir o arquivo. Uma vez executados, os vírus se replicam infectando mais aplicações e são espalhados para outros computadores através da rede, de *e-mails*, compartilhamento de arquivos, entre outros meios.

2.2.1.2 *Worms*

Worms são uma subclasse de vírus, pois também são auto replicantes, porém diferentemente dos vírus não infectam arquivos existentes. Eles se instalam nos computadores e permanecem inativos até o momento adequado para penetrar em outros sistemas através de redes vulneráveis, por exemplo. Como os *worms* não precisam de um hospedeiro, como os vírus, são um tipo de software autônomo, e somada esta característica a de que eles não precisam de ajuda humana para se propagar, os *worms* se tornam *malwares* mais perigosos que os vírus.

2.2.1.3 *Trojans*

Trojans são outro tipo de *malware*, parecem programas legítimos, mas na realidade são arquivos maliciosos disfarçados. Os usuários normalmente são enganados para carregá-los e executá-los em seus sistemas. *Trojans* também são conhecidos por criar aberturas a usuários mal-intencionados acesso ao sistema. Ao contrário dos vírus e *worms*, os *trojans* não se replicam infectando outros arquivos nem fazem cópias de si mesmos, eles se espalham através se alguma interação do usuário, como abrir um anexo de *e-mail* ou baixar e executar um arquivo da Internet.

2.2.1.4 *Ransomware*

Atualmente, o *ransomware* é o mais popular dos *malwares* existentes. Este é um tipo de *malware* que, ao invadir um computador, bloqueia o acesso a ele ou a informações presentes nele, e para conseguir o desbloqueio o usuário precisa pagar, geralmente em *bitcoins* ou alguma outra moeda virtual. Embora não seja uma nova ameaça, evoluiu para se tornar o tipo de *malware* mais rentável da história, e as empresas estão se tornando um alvo de escolha para alguns operadores de *ransomware*. No primeiro semestre de 2016, as campanhas de *ransomware* direcionadas a usuários individuais e empresariais tornaram-se mais difundidas e potentes.[11]

2.2.1.5 *Rootkit*

Rootkit é um conjunto de softwares projetados para ocultar a sua presença ou a de outro software mesmo após infectar o computador do usuário. E permite o acesso de hackers ao sistema para fazer *downloads* e *uploads* de *malwares* a qualquer momento. Ele é capaz de se ocultar utilizando-se de algumas camadas inferiores do sistema operacional, o que o torna quase indetectável pelos softwares *antimalware* comuns.

2.2.1.6 *Spyware*

O *spyware* é um *malware* de espionagem que recolhe informações do sistema após infectá-lo. Este *malware* pode ser utilizado, por exemplo, para criar perfis do usuário, pegar informações bancárias ou pessoais. Um exemplo de *spyware* é o *keylogger*, ele registra as atividades do usuário, senhas utilizadas, entre outros. O *spyware* não necessariamente é ilegal, ele pode ser instalado junto com um aplicativo após o usuário aceitar o termo de licença do mesmo, que pode especificar que este *spyware* será instalado para coletar informações do usuário. Desse modo, os desenvolvedores deste *malware* podem também rentabilizar o programa com a venda de informações estatísticas e distribuir o *malware* gratuitamente.

2.2.1.7 *Backdoor*

Backdoor é um tipo de *malware* que dá acesso ao computador do usuário infectado ignorando os mecanismos normais de autenticação. Ele age em segundo plano e é um dos *malwares* mais

perigosos, pois dá ao atacante o poder de realizar qualquer ação possível no computador infectado, além de espioná-lo, controlar todo sistema e atacar outros computadores. O *backdoor* combina várias ameaças de privacidade e segurança, pois pode conter várias outras características destrutivas de infecção de arquivos, criptografia e outros.

2.2.1.8 *Downloader*

Os *Downloaders* são utilizados para fazer o download e instalar programas maliciosos no computador do usuário infectado através do servidor do atacante.

2.2.2 *Malwares* Empacotados e Ofuscados

Para tornar um arquivo mais difícil de detectar ou analisar, os criadores de *malwares* utilizam frequentemente empacotamento ou ofuscação. Programas legítimos incluem quase sempre muitas strings, porém um *malware* que é empacotado ou ofuscado contém muito poucas strings. Ao varrer um arquivo executável por strings, e o mesmo retornar poucas strings, provavelmente ou o arquivo está ofuscado ou empacotado, sugerindo que pode ser nocivo. Será apresentado métodos de ofuscação baseado em *packers*, criptografia, *blinders* e *joiners*.

2.2.2.1 *Packers*

Packers ou *compressors* atuam comprimindo ou cifrando executáveis originais, buscando dificultar a identificação pelos antivírus e análise estática dos códigos por parte do analista. Porém, enquanto compactados, os programas não desempenham as suas funcionalidades primárias. Assim, uma rotina de descompressão, normalmente localizada ao final do arquivo é necessária antes de serem carregados na memória.[12]

2.2.2.2 *Cryptors*

Criptografar um executável é o objetivo de um *cryptors*, também conhecido como *encryptors* ou *protectors*. Esta técnica possui características similares aos *packers*, porém o resultado é alcançado de uma maneira diferente. Ao invés de compactar o arquivo executável, um *cryptor* simplesmente utiliza de algoritmos criptográficos para ofuscar um arquivo, resultando em dificultar a análise estática e sua identificação por softwares de antivírus ou IDSs (do inglês, *Intrusion Detection System*). Esse tipo de técnica torna difícil a análise através de engenharia reversa. Assim como os *packers*, criptografar um executável torna necessário acoplar ao código uma função inversão de decifração, que funciona em tempo de execução ao programa original, onde todo o processo é feito na memória.[12]

2.2.2.3 *Binders* e *Joiners*

Binders são utilizados para ofuscar códigos com propriedades similares aos *packers* e *cryptors*, com o objetivo de dificultar sua identificação e análise. *Joiners* são utilizados para combinar dois arquivos em um, mantendo a propriedade de execução simultânea, isso é ideal quando um atacante envia um programa lícito e junto insere um *malware* que pode ser instalado pela vítima sem qualquer tipo de interação, esse método é conhecido como instalação silenciosa.[12]

2.3 Código Objeto

Atualmente o processo de programação começa com a elaboração de um código-fonte que é um programa escrito em linguagem de alto nível, ou seja, em alguma linguagem de programação, que é clara para programadores, mas não para os computadores. Assim, é necessário que esses códigos sejam traduzidos para linguagem de máquina para serem submetidos ao computador e processados. Este processo de tradução é denominado compilação, que gera o chamado código objeto.

O código objeto pode ser absoluto (os endereços constantes são endereços reais de memória) ou realocável (os endereços são relativos, tendo como referência o início do programa, e os endereços reais de memória são definidos apenas em tempo de execução)[13]. Se o programa contiver chamadas as funções das bibliotecas (função cosseno, por exemplo) o ligador junta o programa-objeto com a(s) respectiva(s) biblioteca(s) e gera um código-executável (ou programa-executável)[14].

2.3.1 Arquivo Executável

Um programa executável geralmente possui a representação binária das instruções de máquina de um processador específico e chamadas aos serviços do sistema operacional, o que os torna normalmente específicos de um único sistema operacional e de um processador.

2.3.2 Arquivo EXE

Arquivos EXE são usados para instalar e executar em algum Sistema Operacional *Microsoft Windows* programas executáveis. O conteúdo de um arquivo EXE pode incluir códigos e dados de referência que podem ser utilizados para localizar e carregar arquivos associados às aplicações do sistema. Estas podem ser iniciadas a partir de um arquivo com extensão EXE, porém, atualmente estas aplicações são geralmente moduladores e contêm diversos arquivos auxiliares, com extensões como DLL (do inglês, *Dynamic Link Library*).

2.3.3 *Portable Executable*

O formato padrão de armazenamento de dados para arquivos objetos e executáveis do *Microsoft Windows* é o *Portable Executable* (PE), desenvolvido em 1993 pelo Comitê de Padrões de Interfaces de Ferramentas (do inglês, *Toll Interface Standard Committee*). Na estrutura de um PE estão

encapsuladas todas as informações necessárias para que o sistema operacional faça a leitura e execução dos arquivos. O nome *Portable Executable* faz referência justamente à portabilidade da implementação dos PEs em outras plataformas sem necessidade de alteração em seu formato.

O estudo e conhecimento desse tipo de estrutura é de extrema importância para a proteção de softwares, pois no PE encontram-se as informações básicas e fundamentais do funcionamento do programa. Assim, conhecendo bem a estrutura dos PEs é possível desenvolver ferramentas que possam evitar ataques de diversos tipos de softwares e programas mal-intencionados. Para entender melhor o funcionamento do PE, é necessário conhecer o formato deste.

O formato PE inclui bibliotecas dinâmicas de referências para *linking*, exportação e importação de API (do inglês, *Application Programming Interface*) , dados de administração de recursos, dados de TLS (do inglês, *thread-local storage*). A estrutura de dados está ilustrada na Figura 2.8 e contém:

- Cabeçalho MZ do DOS;
- Fragmento do DOS (*stub*);
- Cabeçalho do Arquivo;
- Cabeçalho de Imagem Opcional;
- Diretório de Dados;
- Cabeçalhos das Seções;
- Seções.

2.3.3.1 Cabeçalho DOS

Os primeiros bytes do arquivo PE constituem o cabeçalho DOS. Os dois primeiros bytes deste cabeçalho são formados pelos bytes “MZ” (4D 5A em hexadecimal), que representa a assinatura DOS. "MZ"são as iniciais de Mark Zbikowski, um dos desenvolvedores do MS-DOS [16]. Assim, qualquer arquivo PE deve começar com esta sequência de *bytes*. O cabeçalho DOS é constituído de 64 *bytes*. A Figura 2.9 mostra o arquivo notepad.exe do *Windows* XP SP 2, como exemplo da disposição dos 64 *bytes*.

2.3.3.2 Fragmento do DOS (*Stub*)

O *stub* do DOS é um executável embutido no cabeçalho PE, que é chamado caso o arquivo PE não possa ser executado. Ao carregar o programa na memória, o fragmento *stub* verifica se o sistema é compatível, e caso não for, utiliza esta seção para exibir a mensagem de erro.

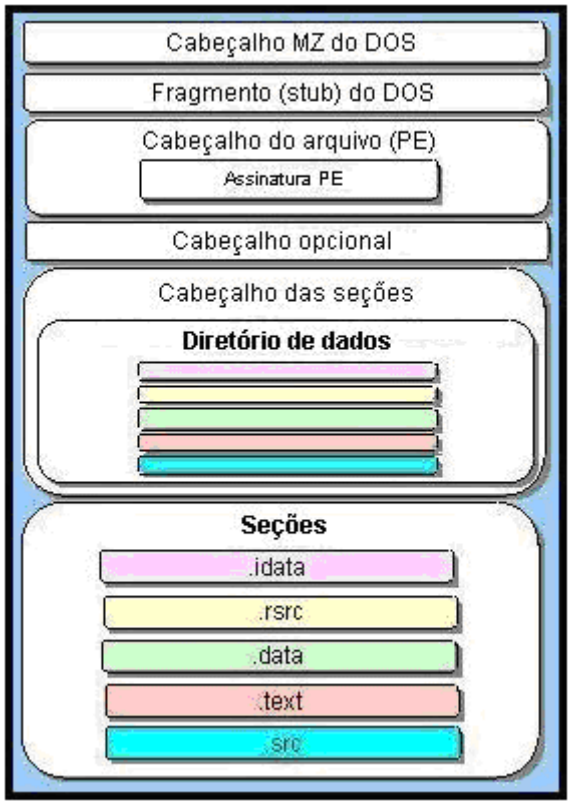


Figura 2.8: Estrutura do formato PE[15].

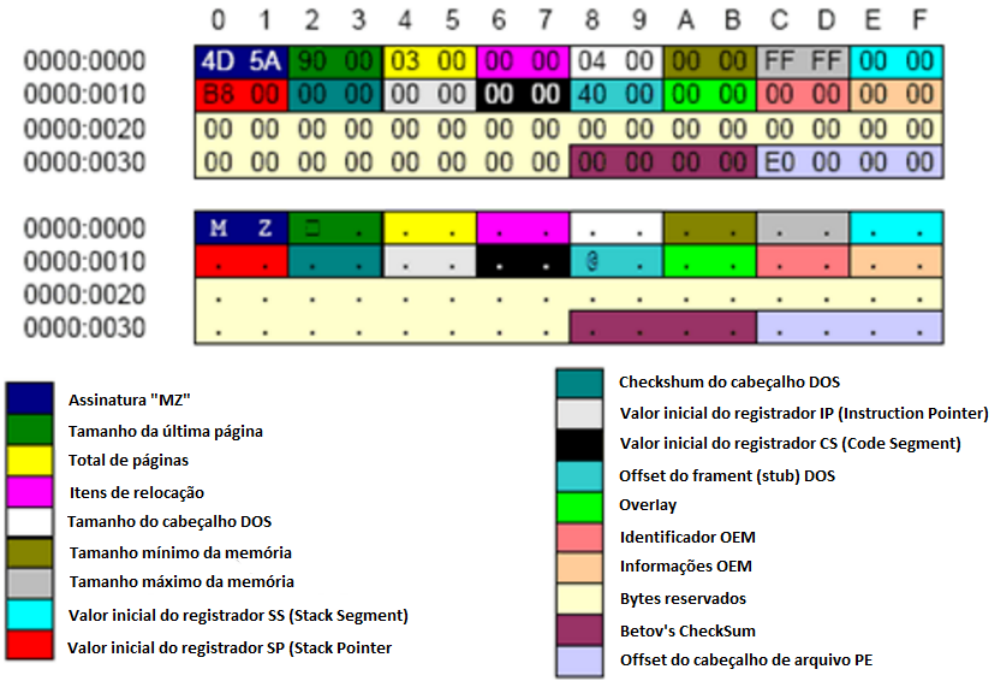


Figura 2.9: Cabeçalho DOS[15].

2.3.3.3 Cabeçalho do Arquivo

O Cabeçalho do Arquivo contém o número, tamanho e onde iniciam as seções, o início da execução do código, entre outras informações necessárias para que o arquivo funcione e que podem variar dependendo da complexidade do PE. A Figura 2.10 ilustra a estrutura deste cabeçalho.

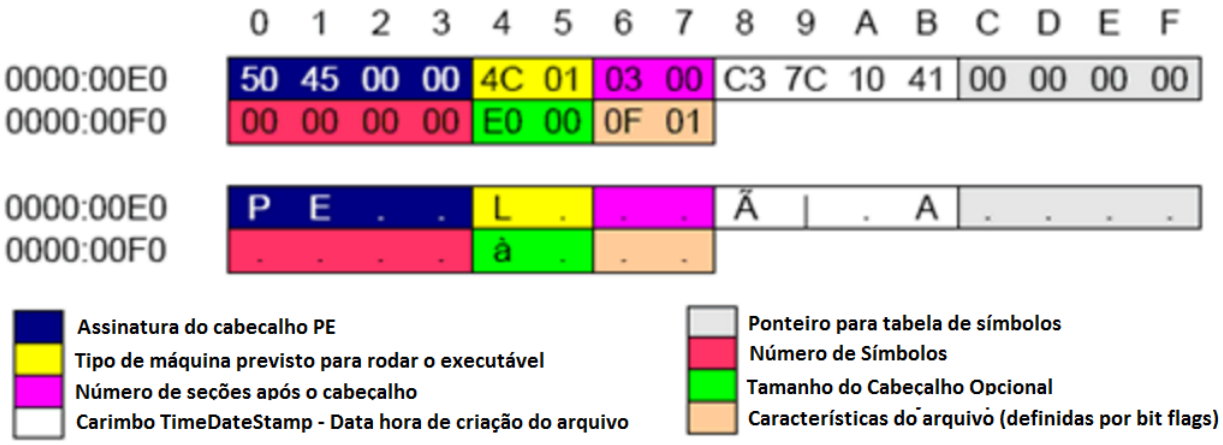


Figura 2.10: Cabeçalho do Arquivo[15].

2.3.3.4 Cabeçalho Opcional

Após o cabeçalho do arquivo vem o cabeçalho opcional que na realidade está em todos PEs e contém informações complementares de como o arquivo deve ser carregado e tratado como, por exemplo, o endereço inicial, a reserva para a pilha (*stack*), o tamanho do segmento de dados, além de um *array* que indica os diretórios de dados que contêm ponteiros para dados presentes nas seções.

2.3.3.5 Cabeçalho das Seções

Os cabeçalhos de seção são as descrições que antecedem as seções e contêm:

1. Um *array* (8 *bytes*) com o nome da seção;
2. Tamanho virtual da seção (4 *bytes*);
3. Endereço físico da seção (4 *bytes*);
4. Tamanho alinhado (4 *bytes*) - tamanho dos dados da seção arredondado para cima para o próximo múltiplo do alinhamento de arquivo;
5. *Offset* do início do arquivo em disco até os dados da seção (4 *bytes*);
6. Ponteiro para remanejamento (4 *bytes*) – apenas para arquivos-objeto;

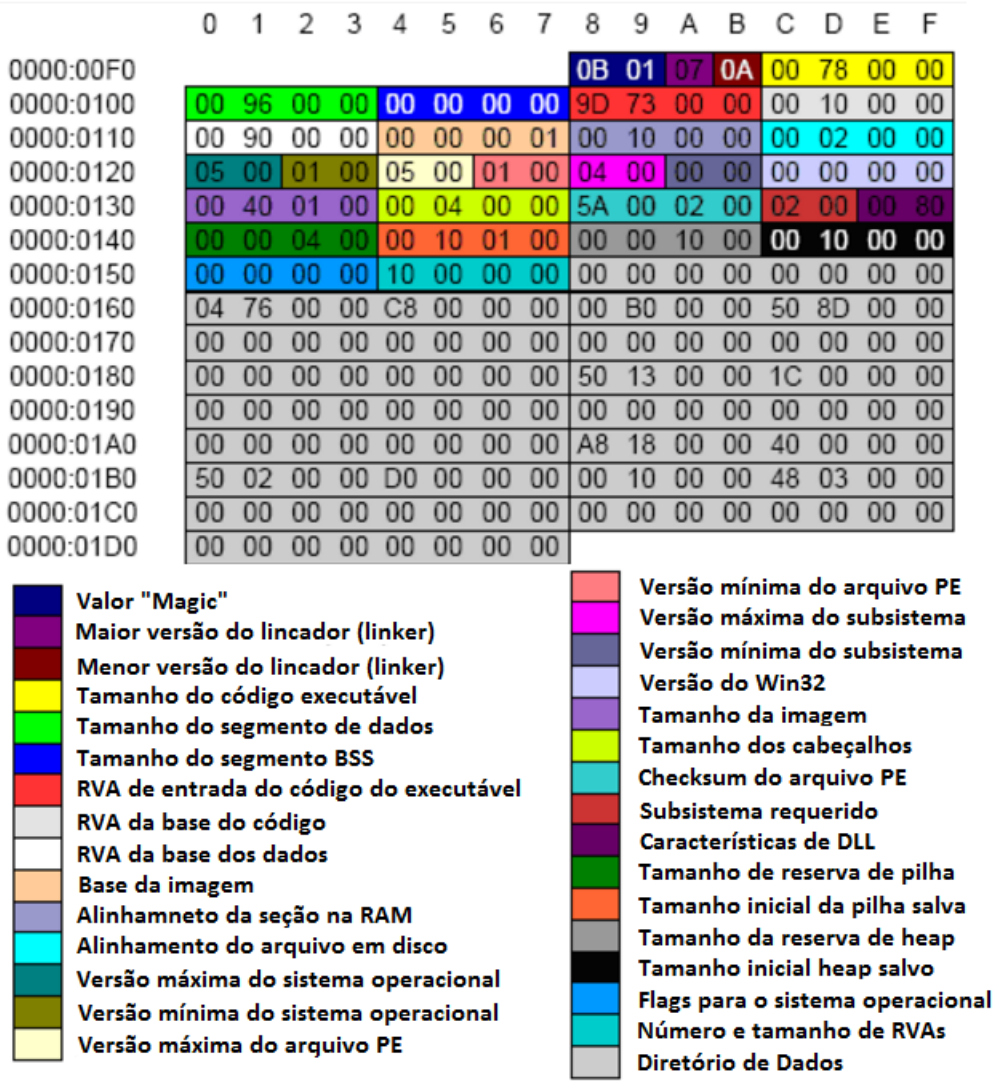


Figura 2.11: Cabeçalho Opcional[15].

7. Ponteiro para números de linha (4 *bytes*) – apenas para arquivos-objeto;
8. Número de remanejamentos (2 *bytes*) – apenas para arquivos-objeto;
9. Quantidade de números de linha (2 *bytes*) – apenas para arquivos-objeto;
10. Características que descrevem como a memória da seção deve ser tratada (4 bytes) (por *bit flag*).[15]

2.3.3.6 Seções

Logo após todos cabeçalhos vêm as seções. Existem vários tipos de seções, dependendo do seu conteúdo, e são nelas que ficam salvas as instruções, recursos (*resources*) e todos os dados e informações do programa propriamente dito. Cada seção possui algumas flags sobre alinhamento,

o tipo de dados que contém, se pode ser compartilhada, etc. Em resumo, é nesta seção onde são gravados os códigos do programa[15].

Algumas seções mais comuns de um EXE:

- Seção de código -*Code Section* (.text ou .code);
- Seção de recursos – *Resource Section* (.rsrc);
- Seção de dados – *Data Section* (.data);
- Seção de exportação – *Export data section* (.edata);
- Seção de importação - *Import data section* (.idata);
- Informações de *debug* - *Debug information* (.debug).[17]

2.3.4 *Extensible Link Format*

Originalmente, o ELF (do inglês, *Executable and Linking Format*) foi desenvolvido pelo USL (do inglês, *UNIX System Laboratories*) e em 1999, foi escolhido como padrão de arquivo binário para os sistemas Unix e similares pelo Projeto 86open. Diferente de outros formatos de arquivos executáveis proprietários, o ELF é mais flexível e extensível, não é específico de um determinado sistema operacional podendo operar em diferentes plataformas e sistemas operacionais. Assim, o uso deste formato facilita a portabilidade para outros sistemas. A estrutura do ELF é similar ao do PE. Cada arquivo ELF tem um cabeçalho e o arquivo de dados que contém:

- Tabela de cabeçalhos de programa (*Program Header Table*) - descreve zero ou mais segmentos. Os segmentos possuem informações usadas pelo sistema operacional para carregar o programa na memória.
- Tabela de cabeçalhos de seção (*Session Header Table*) - descreve zero ou mais seções. As seções possuem dados importantes para a vinculação e a realocação das informações
- Dados do programa - são referenciados pelas entradas da tabela de cabeçalhos de programa ou pelas entradas da tabela de cabeçalhos de seção.

2.4 Funções de *Hashing*

As funções de *hashing* são algoritmos desenvolvidos para verificação de integridade de um arquivo, de modo que dois arquivos diferentes tenham saídas diferentes. O algoritmo é conhecido como uma função unidirecional, onde a entrada variável gera uma saída fixa de tamanho suportado pelo algoritmo, ou seja, com base no resultado de um cálculo não é possível obter o texto de entrada. Para efeito da perícia forense e análise de *malware*, as funções de *hash* são indispensáveis, pois é uma maneira rápida e barata para se comparar aquivos.

Tabela 2.1: Exemplos de funções de *hashing*

Função	**Saída**	**Tamanho**
MD5	58d203edeb444ea93a31cfb067d188aa	128 bits
SHA1	51ba0334b854927fb70c641990eb2cb070af3587	160 bits
SHA256	86122789d7e527dd3d0ac3708242a21d18ee585c0d42ffbef9b1870cb2a76b1e	256 bits

As funções SHA(do inglês, *Secure Hash Algorithm*) e MD5(do inglês, *message-digest*) são funções de hash que atendem os requisitos de segurança provendo uma baixa probabilidade de colisões, que são resultados de dois arquivos diferentes com a mesma saída. Na tabela 2.1 é possível observar exemplos das funções citadas. Para que exista uma maior conformidade de segurança é recomendado que o analista utilize mais de uma função simultaneamente.

As funções de *hashing* facilitam a busca por aquivos, pois se algum artefato é encontrado e seu *hash* calculado, é possível efetuar uma busca por um mesmo resultado independente de nome ou extensão.

2.5 Extração de Dados

Uma atividade fundamental para a segurança de sistemas é a monitoração. Monitorar e armazenar os registros dos eventos ocorridos em um sistema ou rede permitem uma análise posterior desses registros de forma a se procurar por ações suspeitas ou maliciosas, ou mesmo anomalias nos serviços do sistema. Os registros advindos da monitoração servem como dados de entrada para possibilitar a visualização de eventos de segurança e por isso um tratamento especial é indispensável às fontes desses dados.

A necessidade da monitoração vem do fato de que os ataques lançados contra sistemas computacionais deixam rastros, os quais podem ser utilizados para eventualmente identificar a parte invasora, ou para analisar como o ataque foi efetuado. Estes rastros correspondem às atividades realizadas em sistemas e redes alvo para que o ataque tenha sucesso, e armazenam informações diversificadas, tais como a data e a hora do ocorrido, o endereço IP (do inglês, *Internet Protocol*) de origem e de destino, serviços acessados/atacados, o conteúdo do fluxo de dados que resultou no comprometimento de uma rede de computadores ou de um sistema computacional.

2.6 Análise de *Malwares*

Quando um *malware* é descoberto em um sistema, há muitas decisões e ações que devem ser tomadas. Peter Mell e Karen kent[18] propuseram uma metodologia para análise de *malwares* que possui quatro fases principais: preparação, detecção/análise, contenção/erradicação/recuperação e pós incidente. Na tabela 2.2 é feito um breve descritivo de cada fase.

Na Figura 2.12 é apresentado o fluxo de tratamento de um incidente que envolve códigos maliciosos. A detecção é prioritária na condução da análise do incidente, onde a organização deve

Tabela 2.2: Descrição das fases principais na análise de *malwares*

Fase	Descrição
Preparação	As equipes de segurança devem executar medidas preparatórias para garantir que possam responder de maneira eficaz a incidentes de *malwares*.
Detecção e Análise	Detectar, analisar e validar rapidamente um incidente envolvendo *malwares*.
Contenção, Erradicação e Recuperação	Interromper, evitar a propagação do *malwares* e prevenir maiores danos ao sistema. Remover o *malware* dos sistemas infectados. Retornar a funcionalidade e os dados dos sistemas infectados, implantando medidas de contenção temporária.
Pós incidente	Utilizar a base de informações adquiridas para reduzir custos no tratamento dos incidentes e sugerir contra medidas.

possuir um modelo que permite a identificação de um evento de segurança de maneira pró ativa. Os passos da metologia abstrai processos técnicos, mostrando as etapas de forma a permitir a adequação da metologia à organização onde será utilizada.[12]

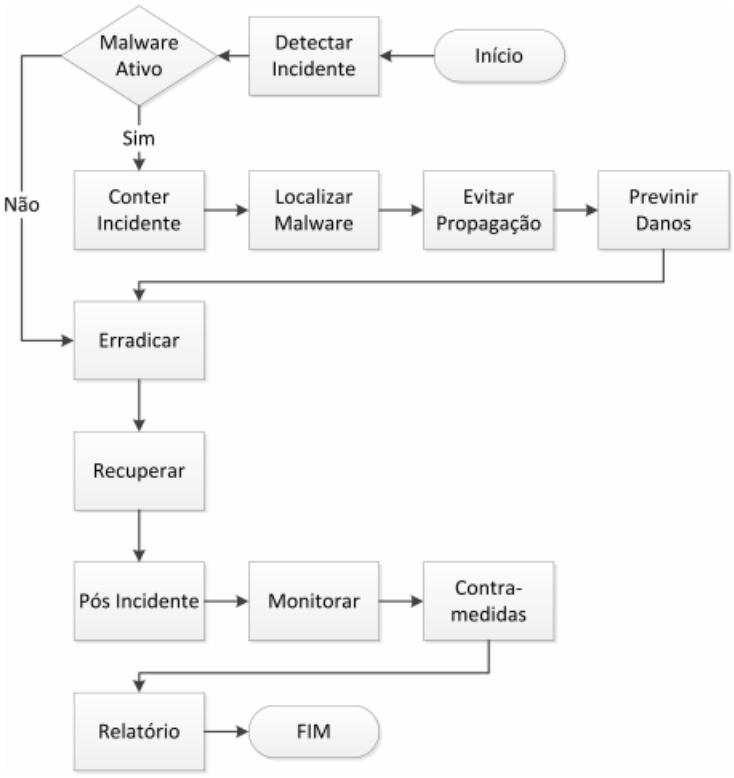


Figura 2.12: Metodologia para Análise de *Malware*[12].

Existem duas abordagens fundamentais para realizar a análise de um *malware*: estática e dinâmica.

2.6.1 Análise Estática

A análise estática consiste em examinar o arquivo sem executá-lo e sem visualizar as instruções do código. Este tipo de análise fornece informações sobre a funcionalidade do *malware*, algo que o identifique, como uma assinatura. Esta análise é geralmente mais eficiente na detecção de *malwares* porque usa informações de alto nível. Ao analisar o arquivo poderão ser encontradas duas situações distintas:

- Quando o código não encontra-se ofuscado, seu código encontra-se em linguagem natural e as informações podem ser coletadas de maneira simples, procurando-as diretamente no código.
- Quando o código foi ofuscado, ou seja, foi utilizada uma técnica de ofuscamento e o código foi "compactado". Nesse caso pouca ou nenhuma informações pode ser obtida sem que o código seja descompactado.

Uma vez que se tem um identificador para o *malware*, é verificado se existe o mesmo *hash* na base de dados do analista, ou se existe similaridade com outro *malware* analisado anteriormente. Caso não sejam encontrados padrões, é iniciada a análise do código. O primeiro passo é obter uma lista de *strings* do código, caso poucas ou nenhuma *string* tenham sido encontradas, é provável que o *malware* esteja utilizando alguma técnica de ofuscamento.

2.6.2 Análise Dinâmica

A análise dinâmica implica em executar o *malware* e observar o seu comportamento no sistema, de modo a remover a infecção, produzir assinaturas, ou ambos. Entretanto, antes de executar o *malware*, deve-se estabelecer um ambiente controlado que permitirá estudar o comportamento do arquivo malicioso sem colocar em risco o sistema operacional ou a rede.

Com relação à execução de um código malicioso, podemos realizá-la de três maneiras distintas:

- Execução manual simples;
- Execução monitorada por meio de um *Instalation Monitor*;
- Execução monitorada por meio de um *API monitor*.

O grande problema da análise dinâmica é que a interpretação dos relatórios é deixada a cargo do usuário que submeteu o exemplar. Além disso, limitações das técnicas utilizadas para interceptar as chamadas de sistema podem levar à evasão da análise por exemplares de *malwares* modernos. Ainda assim, a análise dinâmica é um importante instrumento para prover informações úteis a um usuário.

2.7 Tipologia da Visualização

Munzner[19] definiu três níveis de ações para descrever o objetivo do usuário: análise, busca e requisição.

2.7.1 Ações de Análise

Esta ação pode ser dividida em duas subcategorias, consumir ou produzir, dependendo do objetivo da análise. Na subcategoria de consumir as informações que foram geradas anteriormente são consumidas pelo usuário. Esta subcategoria está dividida em três tipos de ação: descobrir, apresentar e apreciar. Na tabela 2.3 é feito um descrito de cada ação.

Tabela 2.3: Ações de análise da subcategoria consumir

Ação	**Descrição**
Descobrir	Descreve a geração e verificação de hipóteses utilizando a exploração visual, bem como a obtenção de novos conhecimentos sobre os dados apresentados.
Apresentar	Comunicação de informação incisiva ou narrativa baseada nos dados visualizados.
Apreciar	Refere-se ao uso casual ou puro prazer de visualização sem objetivos específicos ou necessidades.

Enquanto na subcategoria produzir, a intenção do usuário é gerar novos materiais ou resultados que serão usados como entrada para outras tarefas. Esta subcategoria está dividida em três tipos de ação: anotar, registrar e derivar. Na tabela 2.4 é feito um descrito de cada ação.

Tabela 2.4: Ações de análise da subcategoria produzir

Ação	**Descrição**
Anotar	Anota a visualização graficamente ou visualmente. Estas anotações são normalmente feitas à mão.
Registrar	Captura ou salva os elementos de visualização selecionados. Em contraste com a ação de anotar, a ação de registrar salva dados da visualização relevantes como um elemento persistente.
Derivar	Produz novos elementos de dados baseados em elementos de dados existentes. Assim, é possível derivar novos atributos a partir de informações existentes ou transformar um tipo de dado em outro.

2.7.2 Ações de Busca

Todas as ações que foram apresentadas na área de análise requerem atividades de busca para os elementos de interesse que são descritos nesta área de nível médio. Munzner dividiu esta área em quatro categorias em que a identificação e a localização dos elementos alvo são conhecidos ou não[19]. Nesta pesquisa, foi utilizado o exemplo de *malware* como o alvo, enquanto as características de *malware* foram usadas como o local. Na tabela 2.5 é feito um descrito de cada ação de busca.

Tabela 2.5: Ações de busca

Ação	Descrição
Investigar	O usuário sabe o que ele está procurando e onde pode ser encontrado (alvo e local são conhecidos). Aplicado à visualização de *malware*, o analista carrega uma amostra específica de *malware* para analisar um conjunto predeterminado de características. Tal ação de pesquisa ajudaria a confirmar uma suspeita ou a investigar certas propriedades da amostra a fim de eventualmente entender seu comportamento.
Localizar	O usuário sabe o que ele está procurando, mas não onde ele tem que procurar (local é desconhecido, mas o alvo é conhecido). Aplicado à visualização de *malwares*, o analista carrega uma amostra de *malware* específico para analisar. No entanto,seus recursos de comportamento relevantes ainda não foram descobertos. Isso significa que o alvo de interesse é conhecido, mas a localização e a características precisam ser localizadas.
Procurar	O usuário não sabe a identidade exata do alvo que está procurando mas sabe o local onde pode ser encontrado. Aplicado à visualização de *malware*, o analista está neste caso interessado em muitas amostras de *malwares* diferentes. O alvo exato é desconhecido, porém ele conhece as características a serem procuradas.
Explorar	O usuário não sabe o que está procurando e também não sabe onde deve procurar (o alvo e o local são desconhecidos). Aplicado à visualização de *malwares*, o analista está interessado em muitas amostras de *malwares* diferentes e não tem um alvo específico em mente. Também não está claro qual das características disponíveis são relevantes, de modo que o local exato a ser observado também é desconhecido.

2.7.3 Ações de Requisição

Uma vez que um (conjunto de) alvos(s) para uma pesquisa é identificado, as informações adicionais serão consultadas como parte desse objetivo de nível inferior. Munzner[19] nomeou três tipos diferentes de consultas que são descritas na tabela 2.6.

Tabela 2.6: Ações de requisição

Ação	Descrição
Identificar	Utiliza apenas um alvo como referência. Se o resultado da busca for um alvo conhecido, a identificação retorna características do alvo.
Comparar	Ações que consideram múltiplos alvos.
Sumarizar	Ações que aplicam para todo os alvos possíveis.

2.8 Trabalhos Relacionados

Várias técnicas de visualização foram propostas para compensar ou ajudar na análise de *malware*. Essas técnicas permitem que os analistas observem visualmente os recursos do malware. Trinius[4] visualizou o comportamento de *malwares* utilizando *Treemap* e *thread graph*, coletando informações sobre as chamadas de API e as operações das ações realizadas em uma *sandbox*. A *Treemap* mostra informações como porcentagem de chamadas de API e informações de seção e o *Thread Graph* mostra os comportamentos cronológicos de um *malware*.

Saxe[20] propôs um sistema para visualizar as relações de sequência de chamadas do sistema. Eles extraíram sequências de chamadas do sistema significativas dos registros de chamadas e construíram uma representação vetorial booleana da estrutura dos arquivos binários de *malwares* e, em seguida, duas interfaces de visualização foram implementadas. A primeira mostra um mapa de similaridade e a segunda mostra similaridades e diferenças entre as amostras selecionadas. As técnicas de visualização baseada na análise estática também foram propostas.

Conti[21] apresentou um sistema de visualização integrado que contém muitas técnicas de visualização gráfica. Seu sistema mostra cada *byte*, a presença de *bytes* e sequências duplicadas de *bytes* contidos dentro de uma amostra. Devido à sobrecarga do algoritmo de plotagem de pontos, eles implementaram o algoritmo simplificado aplicando essas técnicas de visualização.

Nataraj[22] propôs um método para classificar *malware* usando o processamento de imagem. O método proposto representava arquivos binários em imagens de *bitmap* em escala de cinza, digitalizando cada bit em arquivos binários, convertendo cada valor de *bit* em um *pixel* de imagem. Depois de gerar imagens, aplicaram uma técnica de representação abstrata para computar os recursos de textura e classificar o *malware*.

André Gregio[23] apresentou uma estrutura de visualização de eventos comportamentais que permite uma realização mais fácil da cadeia de eventos e detecção rápida de ações interessantes. Além disso, analisou mais de 400 amostras de *malware* de diferentes famílias e mostrou que podem ser classificadas com base na sua assinatura visual. Assim, a possibilidade de diferenciação visual de famílias indica que é possível aplicar, uma técnica automatizada para classificar, agrupar ou extrair dados comportamentais.

Existem diversos trabalhos de pesquisa que usam ferramentas de visualização para auxiliar na análise de dados fornecidos pelos registros relacionados com a segurança. Alguns deles não são abertos ao público, outros não são tão intuitivos de usar. As abordagens atuais de análise visual de *malware* representam vários desafios para o processo analítico que complicam a visualização da imagem. Tendo que visualizar todo o conteúdo em uma única imagem não só complicam o desenvolvimento da ferramenta, mas também podem causar confusão para os analistas. Assim, pretendemos fornecer uma visão geral sobre o estado atual das abordagens de análise visual disponíveis.

Capítulo 3

Métodos Propostos

Neste capítulo é apresentado o cenário utilizado para realizar a análise proposta. Serão utilizadas quatro ferramentas distintas: Binvis.io, Binvis, Binview e Cantor Dust. Todas as ferramentas disponibilizam análise interativa dos arquivos e técnicas baseadas em *pixel*. Contudo, somente o Binvis.io e Binvis são focados nas técnicas baseadas em *pixel*, enquanto o Binview é focado na técnica geométrica e o Cantor Dust em técnica tridimensional. Nas próximas seções será feito um breve descritivo de cada ferramenta assim como um roteiro de uso.

3.1 Ferramentas Utilizadas

Há uma abundância de programas e serviços disponíveis para analisar dados. Ser capaz de visualizar e interagir com dados significa poder interpretá-los de uma maneira diferente com potencial para novas descobertas.

3.1.1 Binvis.io

O criador do Binvis.io, Aldo Cortesi[24], vem experimentando a visualização de dados há vários anos e tem usado a ferramenta para visualizar entropia em arquivos binários, bem como aplicá-la a *malwares*. Com a ferramenta é possível explorar visualmente dados binários, agrupar *bytes* para selecionar características estruturais, alternar entre vários mapeamentos de cores de bytes úteis, incluindo um visualizador de entropia que permite selecionar seções compactadas ou criptografadas e exportar segmentos de dados para análise.

A ferramenta separa o arquivo em intervalos regulares, traduz cada *byte* amostrado para uma cor e escreve o *pixel* correspondente à imagem. Binvis.io disponibiliza duas técnicas para escrever os *pixels*, a *scan* e o *cluster*. Quando é utilizada a *scan*, os *pixels* são escritos linha por linha, serpenteando de um lado para outro para certificar-se que cada *pixel* é sempre adjacente ao seu antecessor. Quando selecionado o *cluster* é utilizada uma construção matemática chamada curva de Hilbert[25] para preservação da localidade. A curva de Hilbert permite uma maneira eficaz de tomar uma matriz unidimensional de bytes, e organizá-los em duas dimensões, de forma que os

bytes que estão próximos uns dos outros em uma única dimensão também estão próximos em duas dimensões. A contrapartida é que a curva de Hilbert não é muito intuitiva visualmente, é difícil coletar informações sobre o deslocamento em um arquivo simplesmente olhando a imagem.

3.1.1.1 Esquemáticos de Cores

A ferramenta disponibiliza quatro diferentes esquemáticos de cores: *byteclass*, *magnitude*, *detail* e *entropy*

O esquemático de cores *byteclass* comprime em algumas classes comuns, conforme a tabela 3.1, todos os 256 valores existente de *byte*. Esta classificação abrange os *bytes* de preenchimento mais comuns, bem como destaca sequências de caracteres, e aglomera todo restante. Na Figura 3.1 é representada a visualização do *malware* DarkComet utilizando o esquemático de cores *byteclass*.

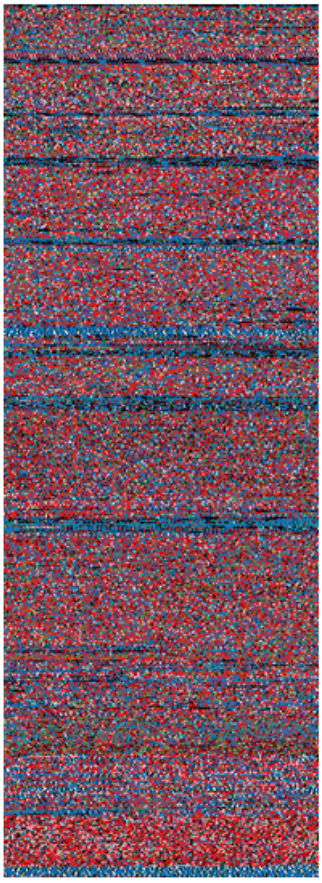


Figura 3.1: Análise do *malware* DarkComet utilizando a ferramenta Binvis.io.

O esquemático de cores *magnitude* varia com os *bytes*, porém apresenta uma luminância calculada para aumentar linearmente. É um sistema de cores intuitivo que pode revelar pequenos detalhes estruturais que não aparecem no esquemático *byteclass*, conforme Figura 3.2.

O esquemático de cores *detail*, que pode ser visualizado na Figura 3.3, atribui uma cor a cada valor de *byte* diferente. Ele tenta maximizar a diferença entre as cores, e ao mesmo tempo manter

Tabela 3.1: Legenda de cores utilizadas pela ferramenta binvis.io quando utilizado o esquemático *byteclass*

Cor	***Byteclass***
	0x00
	Valores inferiores ao ASCII
	ASCII
	Valores superiores ao ASCII
	0xFF

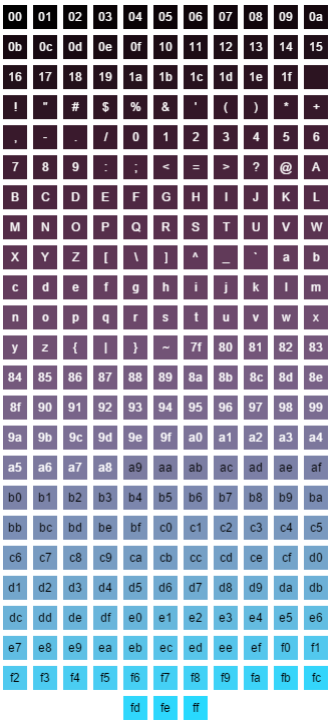


Figura 3.2: Esquemático de cores *magnitude*.

as cores para *bytes* que possuem valores próximos.

O esquemático de cores *entropy* calcula a entropia de Shannon sobre a janela que está em volta do byte em questão, conforme Figura 3.3. Esta visão permite selecionar rapidamente seções cifradas e compactadas.

3.1.1.2 Roteiro de uso do Binvis.io

A utilização é bastante intuitiva e simples, para carregar o arquivo basta clicar no botão "*Open File*"disponibilizado na tela inicial. Após carregado o arquivo irá aparecer um painel de navegação, conforme Figura 3.4, a imagem gerada e um painel de visualização dos hexadecimais, conforme Figura 3.5. No painel de navegação os botões 1, 2 e 3 fazem os ajustes no *zoom*, os botões 4 e 5 permitem escolher respectivamente a forma de escrita e o esquemático de cores a serem utilizados.

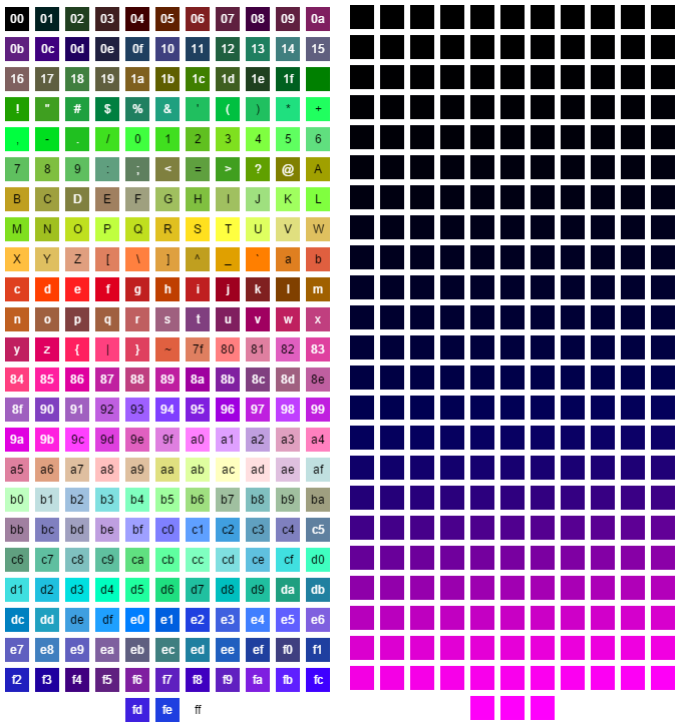


Figura 3.3: Esquemático de cores *detail* e *entropy* respectivamente.

E por ultimo o botão 6 permite salvar a imagem gerada. No painel de visualização dos hexadecimais é possível realizar a leitura dos hexadecimais do arquivo para auxiliar na análise.

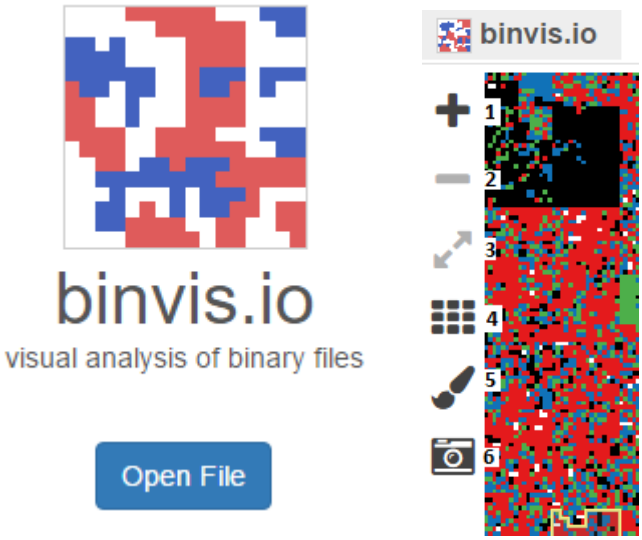


Figura 3.4: Tela inicial e painel de navegação do Binvis.io.

Binvis.io parece simples, porém surpreende por ser eficaz. Se o arquivo analisado for em sua maioria azul, por exemplo, provavelmente é algum tipo de arquivo de texto. Se o arquivo é em sua maioria cores aleatórias, então é provável que seja um arquivo compactado ou empacotado e

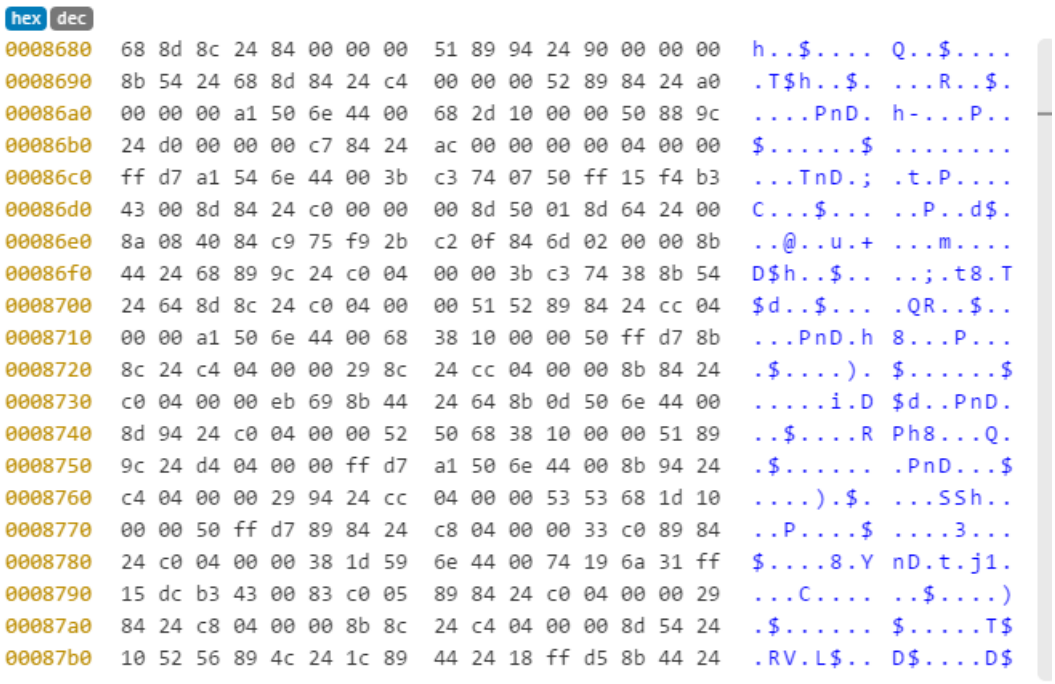


Figura 3.5: Visualização dos hexadecimais em determinada região do Binvis.io.

talvez seja um arquivo de imagem, vídeo ou áudio.

3.1.2 Binvis

BinVis[26] é um projeto baseado em C, desenvolvido pelo Gregory Conti[27], para visualizar estruturas de arquivos binários. O sistema incorpora tanto técnicas de visualização gráfica e textual com objetivo de combinar a funcionalidade das ferramentas de linha de comando e das melhores práticas de editores hex.O sistema inclui diferentes tipos de visualizações que serão descritas nas sessões a seguir.

3.1.2.1 Byteview Visualization

Esta visualização mapeia cada *byte* no arquivo em um *pixel* da imagem. O primeiro *byte* do arquivo está localizado no canto superior esquerdo, na coordenada (1,1), o próximo *byte* é disposto na posição (2,1). A resolução é de 640x480, portanto cada linha irá mostrar 640 *bytes*, permitindo uma visualização máxima de 307,2 *kbytes*. Quando o final da linha é alcançado, a plotagem continua na próxima linha a seguir. Na Figura 3.6 é apresentado um exemplo desta visualização para o *malware* DarkComet.

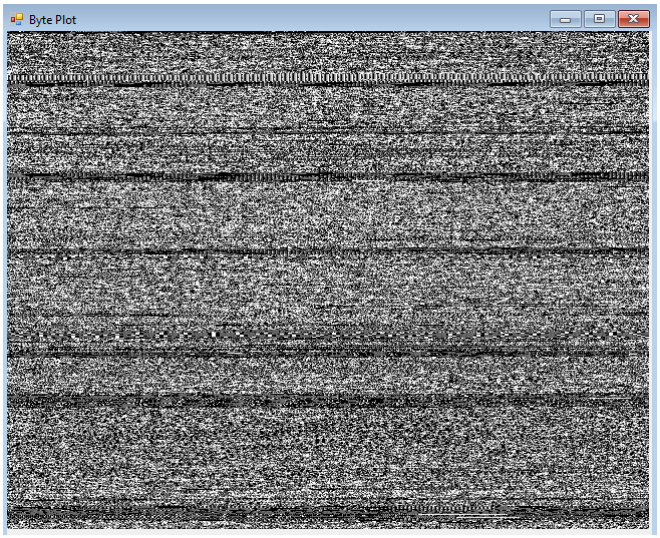


Figura 3.6: Byteview gerado para o *malware* DarkComet.

3.1.2.2 Byte Presence Visualization

Esta visualização consiste de 256 colunas, cada linha mostra a presença e a ausência de valores de *bytes* em determinado intervalo examinado no arquivo. Esta visualização é designada para atuar em conjunto com a *byteview visualization* e cada uma das 480 linhas é representada na linha correspondente desta visualização. Ao projetar essas duas visualizações para agirem em conjunto, um analista é capaz de realizar uma comparação em uma determinada região de interesse. A *byte presence visualization* é particularmente útil para a identificação de regiões de texto contido dentro de um arquivo (visto como barras verticais entre as colunas 32 e 127), para a detecção de valores de *bytes* que mudam regularmente(visto como linhas diagonais, onde o declive é igual a direção e taxa de mudança), para a identificação de regiões de compressão ou de criptografia (visto como uma linha horizontal quase completa), bem como para a detecção do conjunto de caracteres utilizados por um esquema de codificação, tais como a codificação *uuenconding* que utiliza um subconjunto de caracteres ASCII. Na Figura 3.7 é apresentado um exemplo desta visualização para o *malware* DarkComet.

3.1.2.3 *Dot Plot Visualization*

O *dot plot visualization* é uma técnica de visualização poderosa usada por pesquisadores de bioinformática para medir a auto-similaridade. Kaminsky demonstrou que a técnica também é útil para a análise de dados binários, em particular para detectar visualmente sequências repetidas de *bytes* contidos dentro de um arquivo[28]. O gráfico de pontos de Kaminsky funciona criando uma matriz de uma sequência de *bytes* do arquivo. *Pixels* no *display* são iluminados em todos os locais onde os valores dos eixos horizontais e verticais são idênticos. Na Figura 3.8 é apresentado um exemplo desta visualização para o *malware* DarkComet. É importante notar que o algoritmo pode também ser usado para comparar duas sequências de *bytes* diferentes, tais como dois arquivos

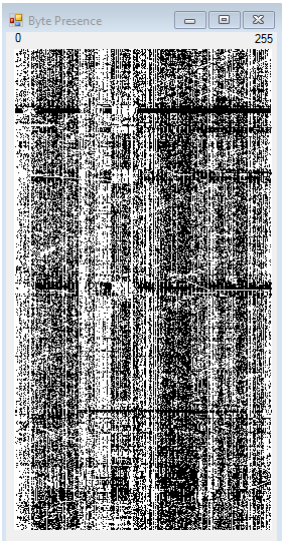


Figura 3.7: Byte Presence gerado para o *malware* DarkComet.

diferentes, e indicar visualmente cada diferença.

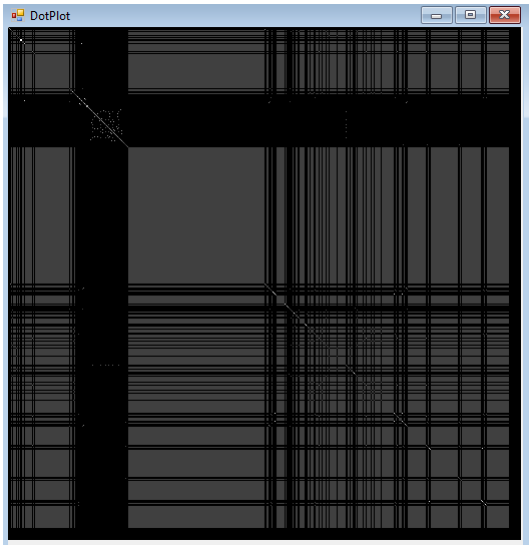


Figura 3.8: Dot Plot gerado para o *malware* DarkComet.

3.1.2.4 Roteiro de uso do Binvis

Uma vez que o Binvis esteja em execução é necessário apenas carregar um arquivo. Quando o arquivo estiver carregado, irá aparecer uma barra de navegação porém sem nenhuma imagem, para escolher uma visualização é necessário ir no menu *view* e escolher a visualização mais adequada para a análise, conforme Figura 3.9.

A navegação pode ser feita de duas maneiras distintas, uma automática que é reproduzida ao clicar no botão *Play* e a outra de forma manual escolhendo a distância de cada salto no arquivo.

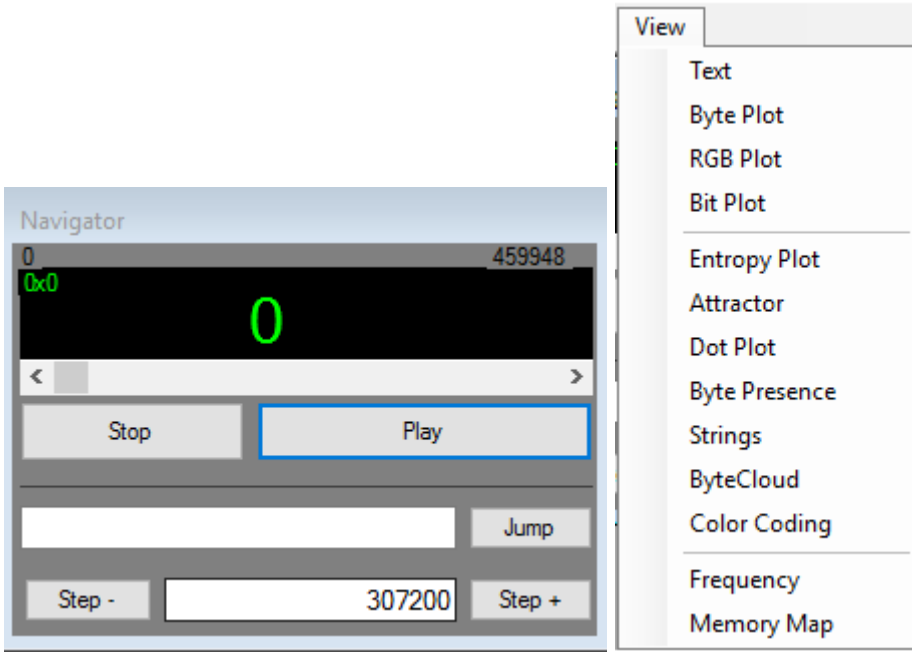


Figura 3.9: Painel de navegação e menu de visualizações do Binvis.

3.1.3 BinView

BinView[29] é um protótipo para visualização de dados binários. Ele pode ser usado para analisar grandes blocos binários e/ou arquivos, e como uma ferramenta de auxílio na engenharia reversa. Na Figura 3.10 é apresentada a visualização para o *malware* DarkComet, utilizando esta ferramenta.

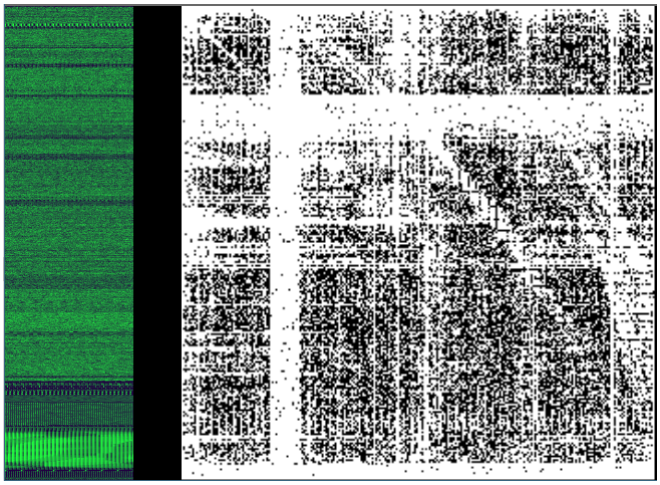


Figura 3.10: Análise do textitmalware DarkComet utilizando a ferramenta BinView.

3.1.3.1 Roteiro de uso do BinView

Para carregar um arquivo para análise a ferramenta contempla o botão "*Open File*", que pode ser visualizado na Figura 3.11, que ao ser clicado permite escolher o diretório e o arquivo a ser analisado.

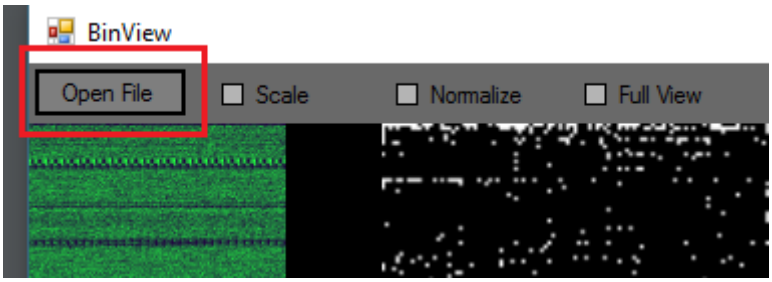


Figura 3.11: Botão *Open File* do BinView.

Quando o arquivo é carregado na ferramenta é disponibilizado um painel de navegação na esquerda, painel de visualização na direita e um painel de funcionalidades na parte superior. A disposição dos painéis pode ser vista na Figura 3.12. O painel de navegação é de fácil utilização, pois ao passar o cursor do mouse em determinada região do arquivo a imagem é disponibilizado no painel de visualização.

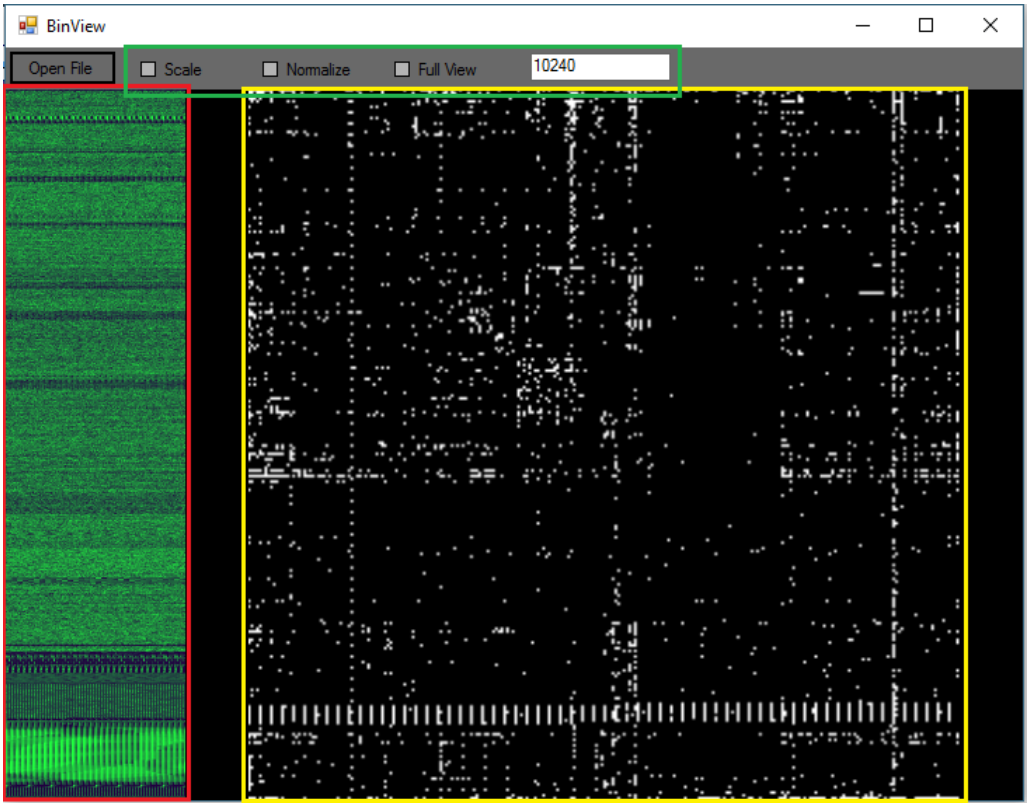


Figura 3.12: Painéis disponibilizados pelo BinView.

Uma das funcionalidades da ferramenta é a escolha do alcance da visualização, por padrão é definido 10240 *bytes*, porém é possível alterar este valor para um que proporcione uma melhor visualização.

3.1.4 Cantor Dust

Cantor Dust é uma ferramenta inovadora de visualização binária interativa para engenharia reversa e análise forense. Ao traduzir informações binárias para abstração visual, o analista pode verificar *megabytes* de dados arbitrários em segundos, procurando padrões de imagem facilmente identificáveis ao invés de sequências de *bytes* familiares.

Mesmo os conjuntos de instruções e os formatos de dados anteriormente não vistos podem ser facilmente localizados e compreendidos através da sua assinatura. Cantor Dust é uma evolução radical do tradicional editor hexadecimal, acelerando drasticamente o processo de análise.

O Cantor Dust contém dezenas de métodos exclusivos para visualizar e analisar dados binários. A versão de demonstração do *software* ilustra apenas a visualização do sistema de 3 variáveis. Com o intuito de permitir maiores visualizações, a ferramenta disponibiliza a visualização em três diferentes tipos de coordenadas: cartesianas, esférias e cilíndricas. Na Figura 3.13 é representada a mesma amostra nas três visualizações.

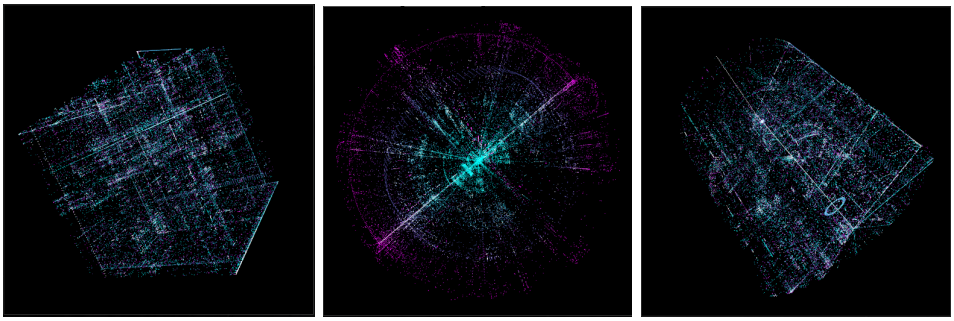


Figura 3.13: Análise do textitmalware DarkComet em coordenadas cartesianas, esféricas e cilíndricas respectivamente.

3.1.4.1 Roteiro de uso do Cantor Dust

A versão de demonstração do *software* vem inicialmente com um exemplo carregado, porém é possível colocar outros arquivos no *software* para serem visualizados. Para carregar um arquivo basta arrastá-lo para a tela do programa.

Os painéis de navegação estão no lado esquerdo da janela e a visualização na direita. Ao clicar e arrastar no painel de navegação mais à esquerda, você pode selecionar uma parte do objeto binário para visualizar.

Para obter opções adicionais é necessário clicar com o botão direito do mouse na visualização.

Capítulo 4

Resultados Experimentais

Neste capítulo serão apresentados os resultados obtidos. Em uma primeira análise, o mesmo tipo de *malware* será analisado em cada uma das ferramentas utilizadas, esta fase será reproduzida para cada tipo de *malware*. Em seguida, será realizada uma comparação das amostras de cada tipo de *malware* quando estas encontram-se empacotadas e quando não se encontram. Por fim, será feita uma análise em relação às diferentes ferramentas.

4.1 Amostras de *Malwares*

Foram coletadas diversas amostras de *Malwares* disponibilizadas pelo Matteo Cantoni em nothink.org[30], porém com o intuito de reduzir a análise, foram separadas apenas três amostras de três classes distintas e outras três amostras de *worms* empacotadas que serão analisadas pelas ferramentas. Na tabela 4.1 é feita a relação das amostras utilizadas. Todas as amostras são do tipo *Portable Executable* 32.

É importante observar que todas as amostras são arquivos *Windows* no formato *Portable Executable* 32 *bits*, pois desse modo facilita-se a comparação entre as amostras e a diferenciação destas em relação a alguns executáveis do *Windows*.

4.2 Análise das Amostras

Inicialmente as imagens obtidas para cada amostra de *malware* serão comparadas com as demais amostras da mesma classe com o intuito de encontrar similaridades. E em um segundo momento será feito uma comparação das amostras entre as ferramentas.

4.2.1 *Worms*

Após analisar as amostras de *worms* no Binvis.io, não foi possível determinar de maneira simples a similaridade entre as amostras, é possível observar que as três amostras possuem regiões

Tabela 4.1: Relação de amostras de *malwares*

	Categoria	**SHA256**
Amostra 1	*Worm*	86122789d7e527dd3d0ac3708242a21d18ee585c0d42ffbef9b1870cb2a76b1e
Amostra 2	*Worm*	5592259600f67f64df3902a53c03c4b5fb91f7f2e2f89429973ff21a152cff72
Amostra 3	*Worm*	7e6f2973c6ac1c108e0fec7e9deee2a697d1bbf370e936c31e2ade094829b440
Amostra 4	*Ransomware*	60b2d7d1cf0d543b5287088fa5f1d594181a128024770fc6cd08cb414a4ab07e
Amostra 5	*Ransomware*	6dd42340d1bf14c90f0a9b6d96f3ca1cde8bf2a1f6a7d3468353752120dfb298
Amostra 6	*Ransomware*	261ea5fad6a80d0884502809b56016eb6acfd89e89b934db22da064bc0f9e955
Amostra 7	*Trojan*	9d3fac012d1f7a6cf3c7c381e6ef4b2c73d4d8d5a3f6a597d2b2837e115c90a0
Amostra 8	*Trojan*	a4a9eed90c8109ff4f2bdcf699ff62acf60e84712d4d80ffb42c54192f41a829
Amostra 9	*Trojan*	5963490f1bd80539f27e1ec78c29540a6e0d47d7783e701418dc5087c6d998c1
Amostra 10	*Packed Worm*	2d6f9a5f114d02a3cb1e3278db4d28d7d2fc79e68d720347a9a730aa355b5720
Amostra 11	*Packed Worm*	697be33fb8a2f193ca55ec5d44dda17c58d7531109c315a932a3dd3581438bfa
Amostra 12	*Packed Worm*	42bbda8a5deb45bb41d9476080f491d24de79415ccd027940ceadec5575e6d2c

bem definidas, como pode ser observado na Figura 4.1. Uma característica importante observada é a existência de preenchimentos com 0's sempre que uma seção inicia e a outra começa, deixando evidente as seções.

As amostras apresentaram apenas as seções de código, dados e recursos, sendo que a seção de dados é puramente preenchimento. Na seção de recurso, nas amostras 1 e 3, é possível observar a presença de uma imagem, enquanto na seção de código é observado uma semelhança entre as amostras 1 e 2.

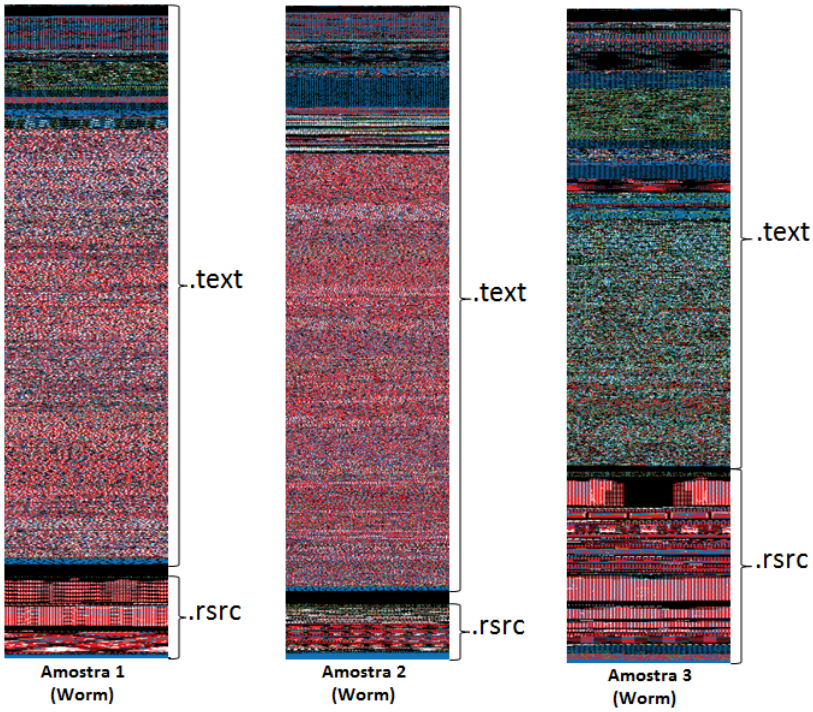


Figura 4.1: Visualização das amostras 1, 2 e 3, respectivamente, utilizando Binvis.io.

No Binvis foram obtidos resultados semelhantes aos obtidos no Binvis.io, como pode ser visto na Figura 4.2. Esse comportamento ocorreu devido ao fato de que as duas ferramentas utilizam técnicas baseadas em *pixel* com diferença apenas no mapas de cores.

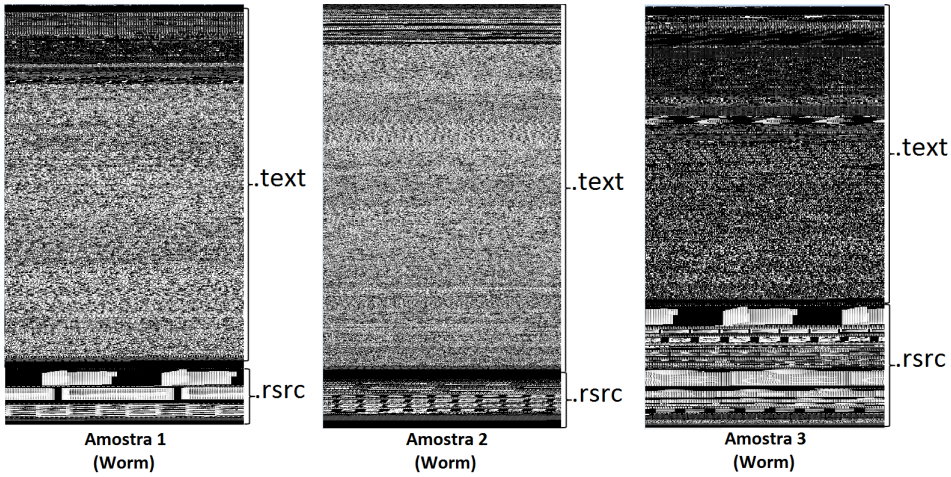


Figura 4.2: Visualização das amostras 1, 2 e 3, respectivamente, utilizando Binvis.

Os resultados obtidos para as amostras utilizando a ferramenta Binview permitiu observar o aparecimento em todas as imagens de um quadrado na região central, uma linha horizontal na parte superior e outra na vertical no canto direito, como pode ser observado na Figura 4.3. Na amostras 1 e 2 as linhas estão mais destacada e o quadrado ofuscado, enquanto na amostra 3 está o inverso. Esta característica evidencia o resultado esperado de que códigos maliciosos de uma mesma família tendem a apresentar semelhanças. Durante a análise não foi possível identificar qual parte do código representa essas duas características.

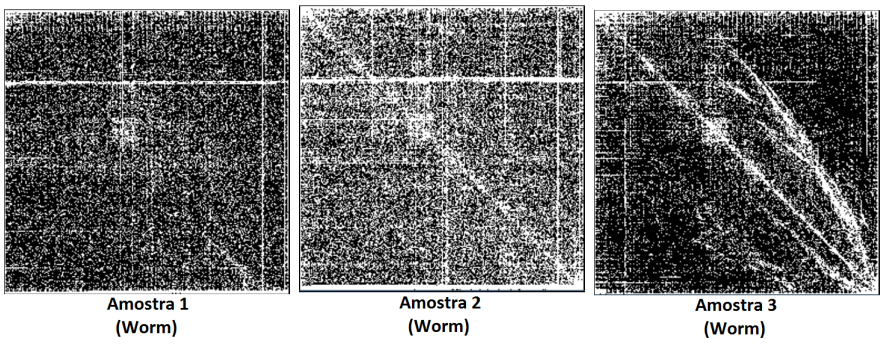


Figura 4.3: Visualização das amostras 1, 2 e 3, respectivamente, utilizando Binview.

A ferramenta Cantor Dust apresentou informações de forma mais clara, pois permite um maior nível de detalhamento. Na Figura 4.4 é visto que as amostras apresentaram resultados semelhantes. Existem regiões bastantes distintas nas imagens, porém três características ficam bem marcadas, o quadrado e uma linha horizontal em uma das faces do cubo e uma linha vertical em outra face. Estas regiões podem ser utilizadas como um ponto de partida para a análise, pois evidenciam um comportamento característico desta categoria de código malicioso.

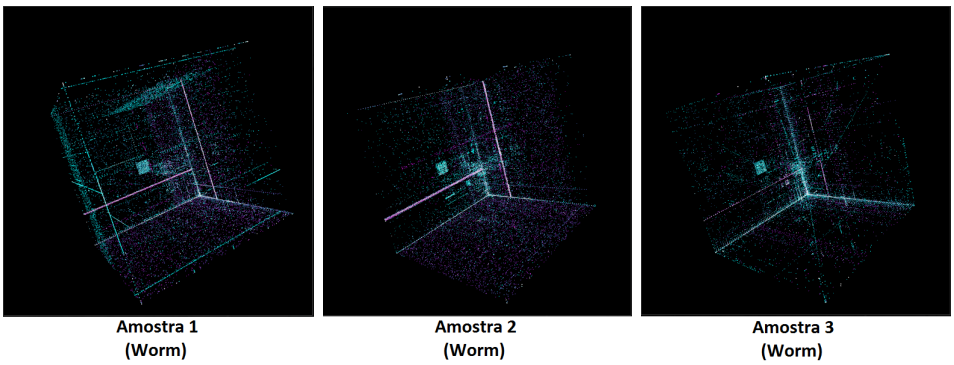


Figura 4.4: Visualização das amostras 1, 2 e 3, respectivamente, utilizando Cantor Dust.

4.2.2 *Ransomware*

Nas Figuras 4.5, 4.6, 4.7 e 4.8 é possível observar que a entropia nas amostras *ransomwares* é bastante elevada, em alguns casos chegando a 7,73 próximo do valor máximo de 8. Como os arquivos parecem um conjunto de *bytes* aleatórios, qualquer análise fica prejudicada. *Malwares* desta família são uma das maiores ameaças na área de segurança e seria possível realizar um trabalho focado na detecção e análise de *ransomware* devido a sua complexidade.

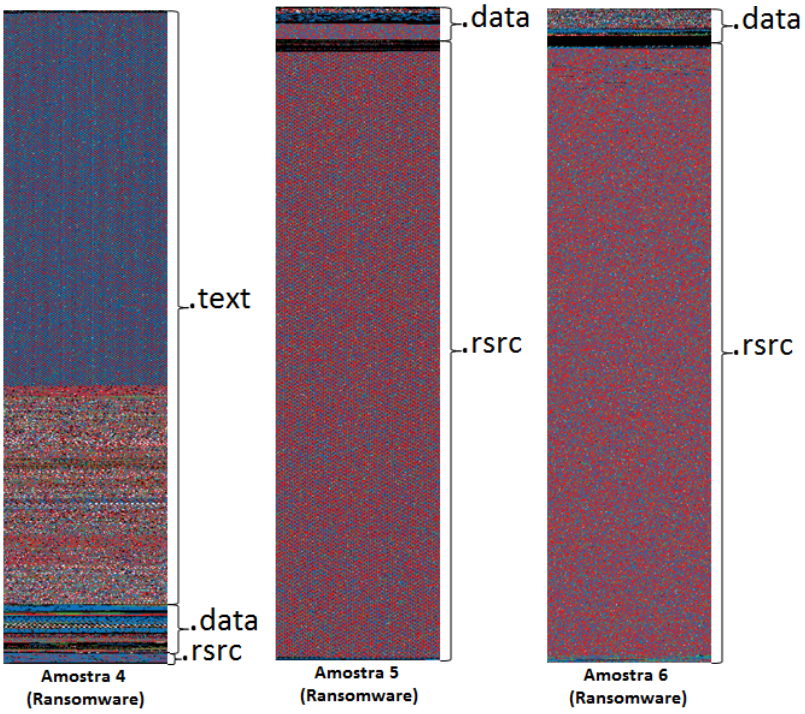


Figura 4.5: Visualização das amostras 4, 5 e 6, respectivamente, utilizando Binvis.io.

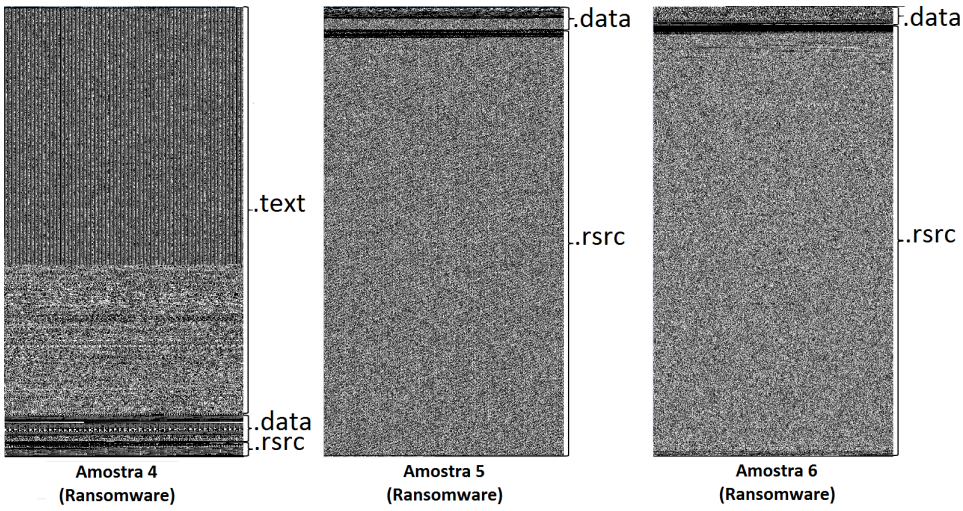


Figura 4.6: Visualização das amostras 4, 5 e 6, respectivamente, utilizando Binvis.

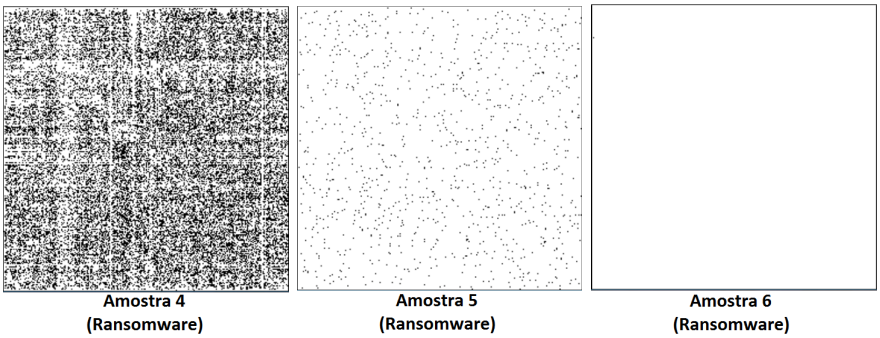


Figura 4.7: Visualização das amostras 4, 5 e 6, respectivamente, utilizando Binview.

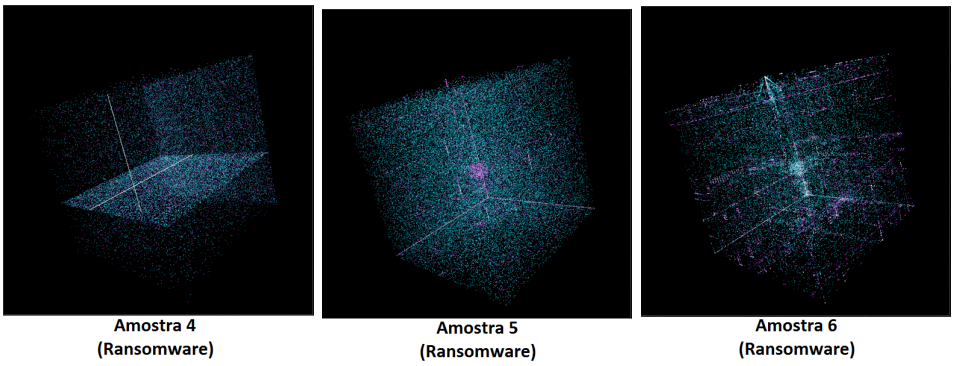


Figura 4.8: Visualização das amostras 4, 5 e 6, respectivamente, utilizando Cantor Dust.

4.2.3 *Trojans*

As amostras de *trojans* selecionadas apresentam estruturas diferentes entre si, fazendo com que as imagens geradas não tivessem semelhança umas com as outras, este fato é possível ser observado nas Figuras 4.9, 4.10, 4.11 e 4.12. Como existe uma grande variedade de *trojans*, para realizar uma

análise mais conclusiva seria necessário pegar uma quantidade muito superior de amostras. Esta nova análise ficará para trabalhos futuros, devido à necessidade de criação de um banco de dados e uma técnica para calcular similaridade entre imagens.

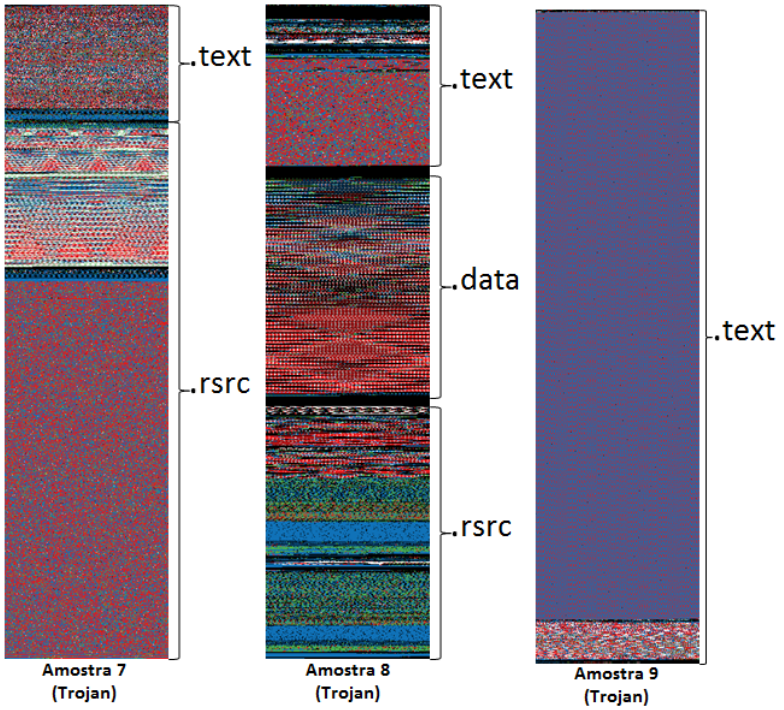


Figura 4.9: Visualização das amostras 7, 8 e 9, respectivamente, utilizando Binvis.io.

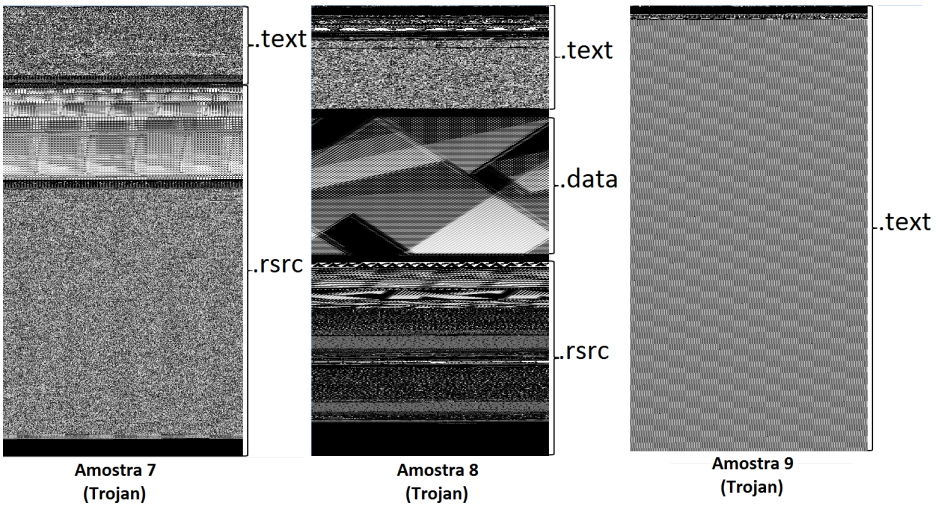


Figura 4.10: Visualização das amostras 7, 8 e 9, respectivamente, utilizando Binvis.

4.2.4 *Malwares* Empacotados

Apesar das amostras estarem empacotadas é possível observar determinada semelhança entre elas, ao analisar essas amostras no Binvis.io é observado que o processo de empacotamento gera

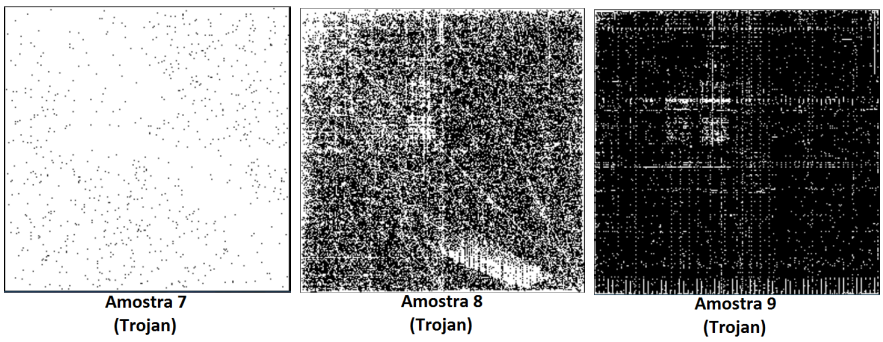


Figura 4.11: Visualização das amostras 7, 8 e 9, respectivamente, utilizando Binview.

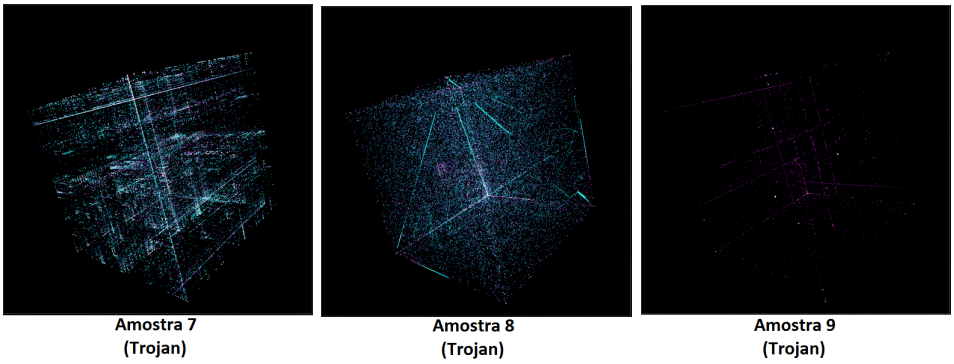


Figura 4.12: Visualização das amostras 7, 8 e 9, respectivamente, utilizando Cantor Dust.

uma saída similar para determinadas seções, mesmo a estrutura de cada amostra sendo diferente. A única diferença aparente entre cada amostra é o tamanho de cada seção que varia de *malware* para *malware*. Comportamento pode ser visto na Figura 4.13.

Ao analisar as três amostras na ferramenta Binview, a saída obtida para elas apresentaram um alto grau de semelhança, como pode ser observado na Figura 4.14. Comportamento que pode ser justificado pela forma de funcionamento do UPX (do inglês, *Ultimate Packer for Executables*) que causou uma maior repetição de determinados valores hexadecimais.

Enquanto nas ferramentas Binvis.io e Binview foi possível identificar alguma semelhança a ferramenta Cantor Dust não foi capaz de gerar uma imagem com algum detalhamento, devido a alta entropia do arquivo, como pode ser observado na Figura 4.15. Apesar da ferramenta não fornecer alguma característica marcante das amostras, o resultado pode ser útil, pois indica para o analista que aquele arquivo está com um alto embaralhamento e que deve ser realizada uma análise mais detalhada.

Com o objetivo de observar melhor o comportamento de amostras empacotadas, as amostras 1, 2 e 3 foram empacotadas utilizando o UPX. Na Figura 4.16 são apresentadas as amostras sem e com empacotamento utilizando a ferramenta Binvis.io. As imagens obtidas demonstram que mesmo após o empacotamento, é mantida a semelhança entre as amostras. A amostra 3 que não possuía similaridade com as demais amostras, após ser empacotada apresentou um alto grau de

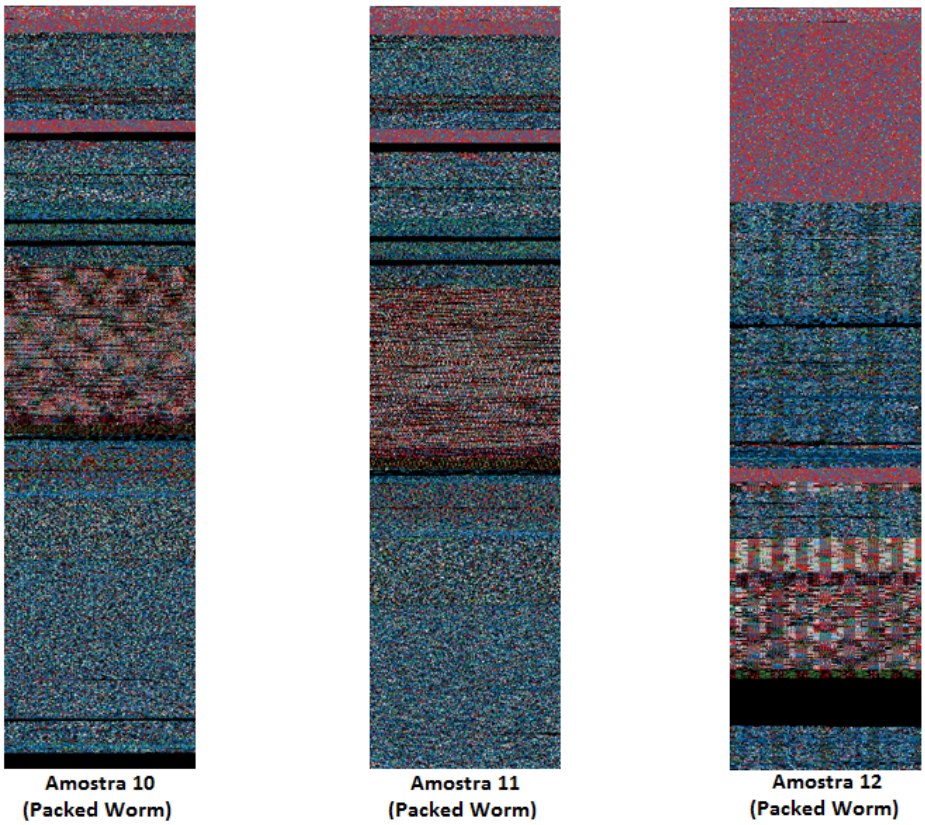


Figura 4.13: Visualização das amostras 10, 11 e 12, respectivamente, utilizando Binvis.io.

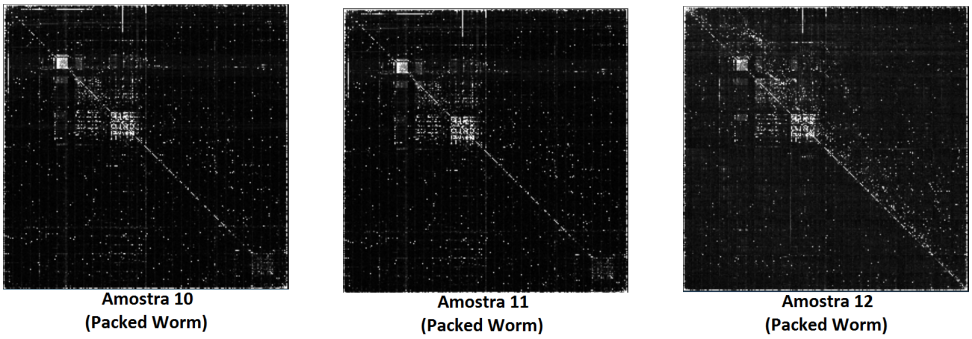


Figura 4.14: Visualização das amostras 10, 11 e 12, respectivamente, utilizando Binview.

similaridade com a amostra 2 empacotada.

Para descartar a hipótese de que a similaridade obtida na Figura 4.16 foi causada pelo empacotamento e que categorias distintas de *malwares* irão apresentar resultados semelhantes, as amostras 2 (*worm*), 4 (*ransomware*) e 8 (*trojan*) foram empacotadas utilizando o mesmo *packer*. Conforme pode ser visto na Figura 4.17, as imagens fornecidas para as amostras não apresentaram similaridades entre si, evidenciando que famílias distintas utilizando o mesmo *packer* apresentarão resultados distintos.

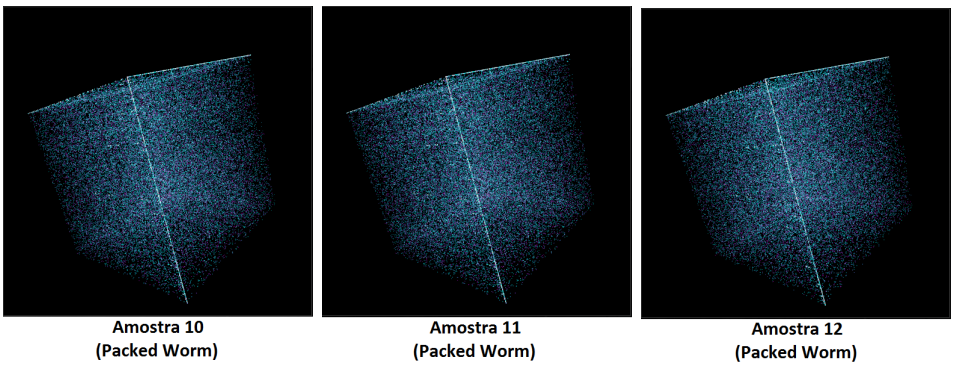


Figura 4.15: Visualização das amostras 10, 11 e 12, respectivamente, utilizando Cantor Dust.

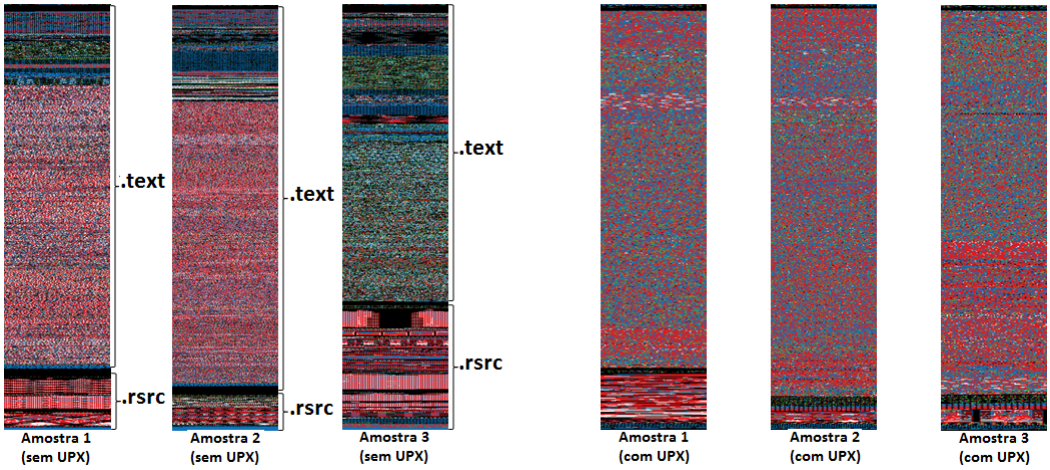


Figura 4.16: Visualização utilizando a ferramenta Binvis.io das amostras 1, 2 e 3, respectivamente, sem (à esquerda) e com (à direita) empacotamento.

4.3 Comparativo entre Ferramentas

Todas as ferramentas utilizadas possuem uma análise visual por *pixel*, a diferença entra elas é que algumas são dedicadas para este tipo de técnica enquanto as outras utilizam ela apenas como base. Na Figura 4.18 é possível observar que a imagem gerada em cada ferramenta é semelhante, diferenciando apenas o nível de detalhamento presente em cada uma devido a seleção de cores.

Apesar de possuírem métodos de análise diferentes, algumas ferramentas apresentaram resultado semelhante para determinadas amostras, como pode ser observado nas Figuras 4.19 e 4.20. Para essas amostras em questão, as duas ferramentas apresentaram o quadrado e as duas linhas. Para a amostra 3, em ambas ferramentas, as linhas ficam menos visíveis e o quadrado fica mais destacado. Este comportamento pode facilitar durante a investigação de um novo código malicioso, pois será possível comparar imagens geradas entre diferentes ferramentas.

A ferramenta Binvis.io demonstrou ser eficiente em identificar as seções de um arquivo, como mostrado nas Figuras 2.7 e 2.8. Ao apresentar uma codificação de cores customizável é possível

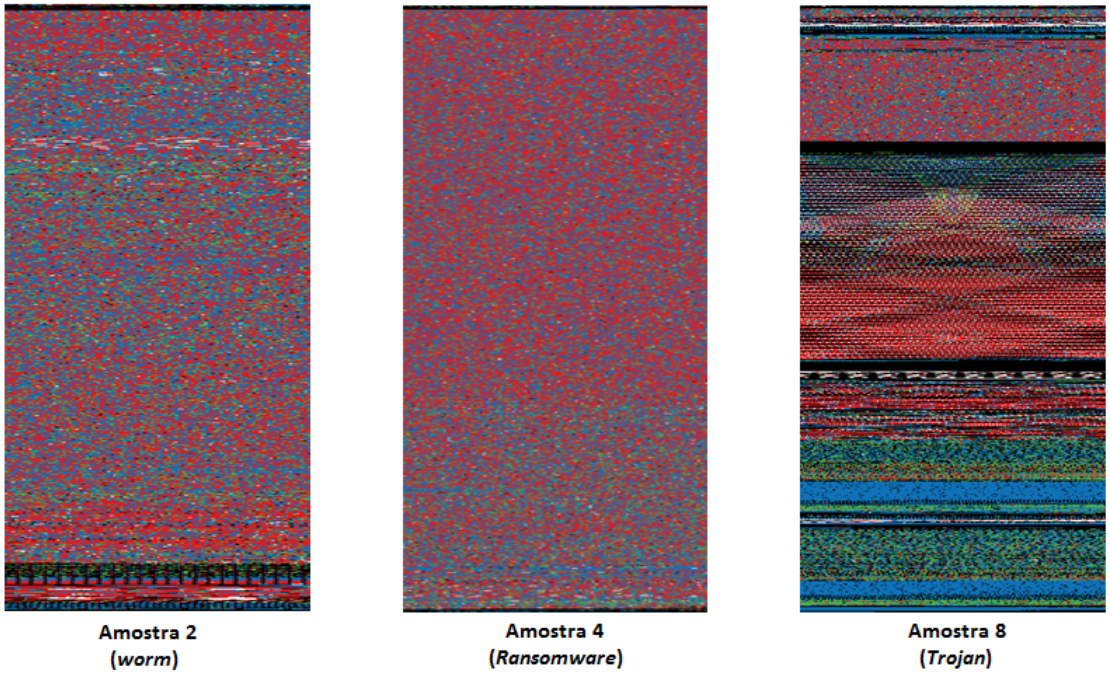


Figura 4.17: Visualização utilizando a ferramenta Binvis.io das amostras 2, 4 e 8, respectivamente, empacotadas com UPX.

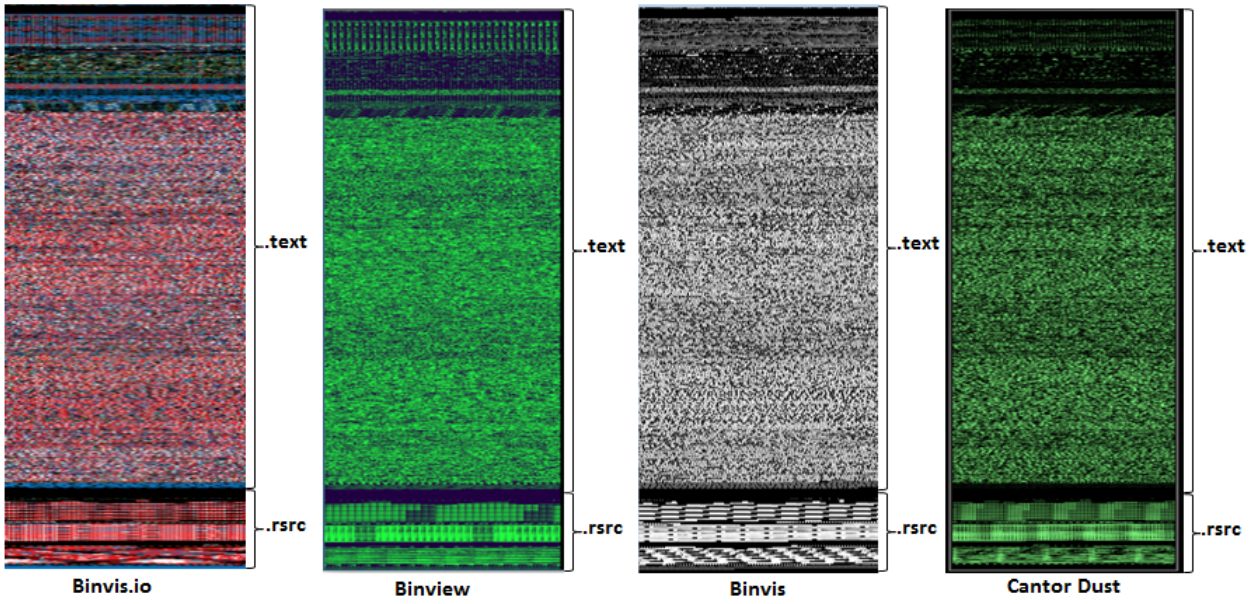
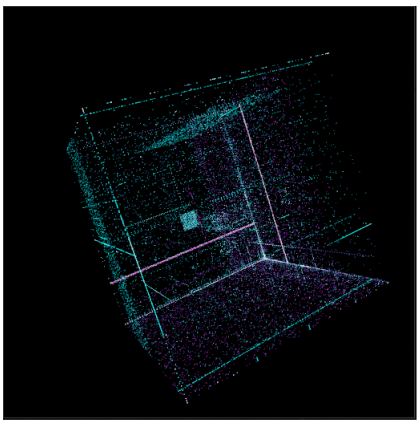
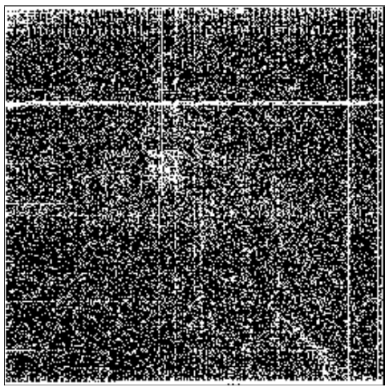


Figura 4.18: Comparativo entre as imagens geradas pelas técnicas baseadas em *pixel* de cada ferramenta para a amostra 1.

adequá-la à necessidade do analista. As ferramentas Binview e Cantor Dust apresentaram uma capacidade em destacar características específicas nos executáveis analisados, o que torna estas duas ferramentas fortes candidatas para serem utilizadas na detecção de *malwares*.

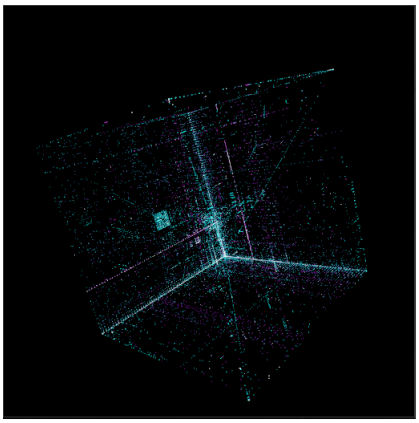


Amostra 1
(Worm)

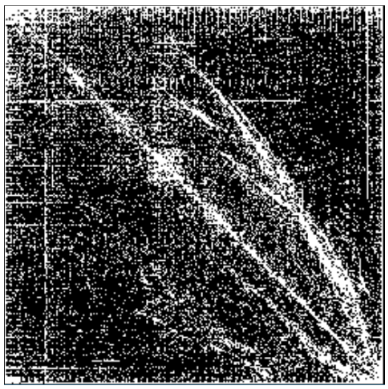


Amostra 1
(Worm)

Figura 4.19: Comparativo entre as ferramentas Cantor Dust e Binview para a amostra 1.



Amostra 3
(Worm)



Amostra 3
(Worm)

Figura 4.20: Comparativo entre as ferramentas Cantor Dust e Binview para a amostra 3.

A técnica de visualização baseada em *pixel* demonstrou apresentar resultados semelhantes em todas as ferramentas, isso é devido ao fato de que essa técnica realiza a leitura do arquivo e faz a tradução *byte* a *byte* por um *pixel*. Caso o arquivo respeite a estrutura de um PE e possua as seções bem definidas é possível detectar semelhança e padrões entre as amostras.

Capítulo 5

Conclusões

Neste trabalho, foram utilizadas quatro ferramentas de análise visual, Binvis.io, BinVis, Binview e Cantor Dust. Foram separadas 12 amostras de *malwares* diferentes e cada amostra foi analisada em todas ferramentas. Com as imagens obtidas, foram verificadas as semelhanças entre as famílias de *malwares* e entre as ferramentas . Neste capítulo são apresentadas a conclusões obtidas a partir da análise realizada no capítulo anterior.

A abundância de informações geradas pelos sistemas de análise pode ser visualizada para auxiliar um analista a compreender melhor determinados padrões. Este trabalho demonstrou efetividade em apresentar similaridades na estrutura interna de um *malware* e entre as famílias de *malwares*. Dependendo da técnica escolhida e os parâmetros utilizados, é possível identificar uma amostra de um *malware* desconhecido, com base nos resultados obtidos anteriormente.

Desenvolvedores de *malwares* modificam pequenas partes do código fonte original para produzir uma nova variante, detectar esta modificação analisando o hexadecimal pode ser complicado e pode conduzir a falhas no processo de detecção e a base de assinaturas requer constante atualização para inserir as variantes recém-criadas. Porém imagens podem capturar pequenas mudanças e manter a estrutura do arquivo facilitando o trabalho do analista na detecção de novas variantes.

Muitas famílias de *malwares* implementam criptografia ou ofuscação do código fonte, o que permite uma identificação mais fácil, pois variantes da mesma família ao serem empacotadas continuarão a exibir similaridade com as demais. O fato de o código estar ofuscado também facilita a identificação, pois indica que o autor quer dificultar a verificação do arquivo por muitas vezes podendo ser um código malicioso. Dependendo do método de ofuscação utilizado a análise pode ficar prejudicada devido ao alto nível de entropia.

Com os resultados obtidos, foi possível perceber a necessidade de existência de uma base de dados sólida para realizar as devidas comparações. Assim como a análise estática, a caracterização de *malwares* como imagens fornecem mais informações sobre a estrutura e não fornecem muitas informações sobre o comportamento do *malware*, devido ao fato destas técnicas não monitorar a execução do código malicioso. Quando combinada com uma técnica de análise dinâmica, a análise visual pode aumentar a eficiência de sistemas de classificação de *malwares*.

5.1 Trabalhos Futuros

Como trabalhos futuro após o desenvolvimento desse trabalho, é sugerido a extensão da análise para *malwares* de outros sistemas operacionais, a elaboração de técnica para calculo de similaridade entre imagens e a implementação de uma interface web, integrando ferramentas de visualização e análise de *malwares*, para detecção e classificação de códigos maliciosos automática.

Referências Bibliográficas

[1] JUNIPER Research. Acessado em Outubro de 2016. Disponível em: <https://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020>.

[2] NOKIA Malware Report. Acessado em Outubro de 2016. Disponível em: <http://company.nokia.com/en/news/press-releases/2016/09/01/nokia-malware-report-shows-surge-in-mobile-device-infections-in-2016>.

[3] AVTEST Malware Statistics. Acessado em Outubro de 2016. Disponível em: <https://www.av-test.org/en/statistics/malware/>.

[4] TRINIUS, P.; HOLZ, T.; GÖBEL, J.; FREILING, F. C. Visual analysis of malware behavior using treemaps and thread graphs. In: IEEE. *Visualization for Cyber Security, 2009. VizSec 2009. 6th International Workshop on*. [S.l.], 2009. p. 33–38.

[5] HOW to visualize data with cartoonish faces ala Chernoff. Acessado em Maio de 2016. Disponível em: <http://flowingdata.com/2010/08/31/how-to-visualize-data-with-cartoonish-faces>.

[6] O que é um dendograma. Acessado em Maio de 2016. Disponível em: <http://support.minitab.com/pt-br/minitab/17/topic-library/modeling-statistics/multivariate/item-and-cluster-analyses/what-is-a-dendrogram/>.

[7] KARIM, M. E.; WALENSTEIN, A.; LAKHOTIA, A.; PARIDA, L. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, Springer, v. 1, n. 1-2, p. 13–23, 2005.

[8] JUNG API. Acessado em Outubro de 2016. Disponível em: <http://jung.sourceforge.net/site/jung-api/>.

[9] GRÉGIO, A. R. A.; FILHO, B. P. de C.; MONTES, A.; SANTOS, R. Técnicas de visualização de dados aplicadas à segurança da informação. *Campinas, São Paulo*, 2009.

[10] SIKORSKI, M.; HONIG, A. *Practical malware analysis: the hands-on guide to dissecting malicious software*. [S.l.]: no starch press, 2012.

[11] CISCO 2014 Annual Security Report. Acessado em Maio de 2016. Disponível em: <http://www.cisco.com/web/offers/lp/2014-annual-security-report/index.html>.

[12] MELO, L. P. de; AMARAL, D. M.; SAKAKIBARA, F.; ALMEIDA, A. R. de; JR, R. T. de S.; NASCIMENTO, A. Análise de malware: Investigaçao de códigos ma-liciosos através de uma abordagem prática. *SBSeg*, v. 11, p. 9–52, 2011.

[13] CONVERSOES de linguagens. Acessado em Outobro de 2016. Disponível em: <http://www.diegomacedo.com.br/conversoes-de-linguagens-traducao-montagem-compilacao-ligacao-e-interpretacao/>.

[14] CONCEITOS Básicos de Programação. Acessado em Outubro de 2016. Disponível em: <http://www.inf.pucrs.br/ pinho/LaproI/ConceitosBasicos/ConceitosBasicos.htm>.

[15] WINDOWS Portable Executable. Acessado em Outubro de 2016. Disponível em: <http://www.devmedia.com.br/windows-portable-executable/857>.

[16] PIETREK, M. Inside windows-an in-depth look into the win32 portable executable file format. *MSDN magazine*, San Francisco, CA: CMP Media Inc., c2000-, v. 17, n. 2, 2002.

[17] ESTRUTURA Basica de um Portable. Acessado em Outubro de 2016. Disponível em: <http://lab-infor.blogspot.com.br/2012/06/artigoestrutura-basica-de-um-portable.html>.

[18] MELL, P.; KENT, K.; NUSBAUM, J. *Guide to malware incident prevention and handling: Recommendations of the National Institute of Standards and Technology*. [S.l.]: US Department of Commerce, National Institute of Standards and Technology, 2005.

[19] BREHMER, M.; MUNZNER, T. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, IEEE, v. 19, n. 12, p. 2376–2385, 2013.

[20] SAXE, J.; MENTIS, D.; GREAMO, C. Visualization of shared system call sequence relationships in large malware corpora. In: ACM. *Proceedings of the ninth international symposium on visualization for cyber security*. [S.l.], 2012. p. 33–40.

[21] CONTI, G.; DEAN, E.; SINDA, M.; SANGSTER, B. Visual reverse engineering of binary and data files. In: *Visualization for Computer Security*. [S.l.]: Springer, 2008. p. 1–17.

[22] NATARAJ, L.; KARTHIKEYAN, S.; JACOB, G.; MANJUNATH, B. Malware images: visualization and automatic classification. In: ACM. *Proceedings of the 8th international symposium on visualization for cyber security*. [S.l.], 2011. p. 4.

[23] GRÉGIO, A. R. A.; BARUQUE, A. O. C.; AFONSO, V. M.; FILHO, D. S. F.; GEUS, P. L. de; JINO, M.; SANTOS, R. D. C. dos. Interactive, visual-aided tools to analyze malware behavior. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2012. p. 302–313.

[24] ALDO Cortesi. Acessado em Outubro de 2016. Disponível em: <https://corte.si/index.html>.

[25] FERNANDES, D. Curvas de hilbert: agrupamento em espaços bidimensionais. *IV Jornadas de Informática da Universidade Aberta*, Universidade Aberta, p. 1–21, 2014.

[26] WELCOME to the BinVis project. Acessado em Outubro de 2016. Disponível em: <https://code.google.com/archive/p/binvis/>.

[27] GREGORY Conti. Acessado em Outubro de 2016. Disponível em: <http://www.gregconti.com/>.

[28] BLACK Ops 2006. Acessado em Maio de 2016. Disponível em: <https://dankaminsky.com/slides/dmk-blackops2006-ccc.ptt>.

[29] BINVIEW. Acessado em Maio de 2016. Disponível em: <https://github.com/russlank/BinView>.

[30] NOTHINK Malware Archives Download. Acessado em Outubro de 2016. Disponível em: <http://www.nothink.org/honeypots/malware-archives/>.