Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Luiz Rodrigo Gomes de Oliveira

Aplicação de algoritmos de aprendizado de máquina na unificação das técnicas de análise estática e dinâmica para classificação de *ransomware*

São Paulo

2018

Luiz Rodrigo Gomes de Oliveira

Aplicação de algoritmos de aprendizado de máquina na unificação das técnicas de análise estática e dinâmica para classificação de *ransomware*

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.
Área de Concentração: Infraestrutura Computacional.

Data da aprovação: _____/_____/_____

_____
Prof. Dr. Anderson A. A. da Silva (Orientador)
Mestrado Engenharia de Computação

Membros da Banca Examinadora:

Prof. Dr. Anderson Aparecido Alves da Silva (Orientador)
Mestrado Engenharia de Computação

Prof. Dr. Marcelo Teixeira de Azevedo (Membro)
UNIESP – União das Instituições Educacionais de São Paulo

Prof. Dr. Eduardo Takeo Ueda (Membro)
Mestrado Engenharia de Computação

Luiz Rodrigo Gomes de Oliveira

Aplicação de algoritmos de aprendizado de máquina na unificação das técnicas de análise estática e dinâmica para classificação de *ransomware*

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.
Área de Concentração: Infraestrutura Computacional.

Orientador: Prof. Dr. Anderson A. A. da Silva

São Paulo

Maio/2018

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT

O48a **Oliveira, Luiz Rodrigo Gomes de**
Aplicação de algoritmos de aprendizado de máquina na unificação das técnicas de análise estática e dinâmica para classificação de ransomware. / Luiz Rodrigo Gomes de Oliveira. São Paulo, 2018.
73p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Infraestrutura Computacional.

Orientador: Prof. Dr. Anderson A. A. da Silva

1. Detecção de malware 2. Ransomware 3. Análise estática 4. Análise dinâmica 5. Support Vector Machine – SVM 6. Naive Bayes 7. Árvore de decisão 8. Tese I. Silva, Anderson A. A. da, orient. II. IPT. Coordenadoria de Ensino Tecnológico III. Título

18-56 CDU 004.492(043)

Bibliotecário responsável: Maria Darci Cornellas Narciso – CRB 8/3569

DEDICATÓRIA

Às pessoas que ajudaram a tornar realidade essa conquista: mãe, pai, esposa e filhos.

AGRADECIMENTOS

Quero agradecer a minha esposa, amiga e companheira Ana Paula Moreira de Oliveira, por ter incentivado esta caminhada, pelas incontáveis leituras e revisões de texto e, principalmente, pela paciência que teve. A minha filha Maria Eduarda por compreender, ou não, minha indisponibilidade quando queria brincar. E a minha recém-motivação, meu filho(a) que ainda não nasceu mas já colabora trazendo felicidade e incentivo.

Aos meus pais, sr. Luiz Delfino e Sra. Ida Gomes, pelo incentivo, motivação, educação e esforços não medidos. Ao meu pai, agradeço imensamente pelas constantes, quase que semanal, leitura e revisão deste trabalho.

Deixo minha imensa gratidão ao meu orientador Prof Dr. Anderson A. A. da Silva, por ter acreditado em mim e neste projeto. Sua valiosa orientação foi de grande contribuição e motivação a esta dissertação. Agradeço-o imensamente.

Por fim, agradeço a Deus por me dar saúde para poder realizar mais uma conquista.

RESUMO

Combater r*ansomware,* programa que bloqueia e/ou criptografa o dispositivo da vítima com objetivo de pedir um resgate pelos arquivos criptografados ou pela tela bloqueada, é uma tarefa difícil diante da rápida disseminação e mudanças que os desenvolvedores aplicam às técnicas de ofuscamento e evasão dos programas *anti-malwares*. Um dos incentivos ao desenvolvimento deste tipo de *malware* é que parte das vítimas optam em pagar pelo resgate das informações devido à ausência de práticas de segurança da informação. Este trabalho propõe a unificação de dois métodos usados na análise e detecção de *malware*, abordando especificamente *ransomware*: as análises estática e dinâmica. A proposta é usar a integração das técnicas para criar um arquivo que contém tanto as características de um *ransomware* como as características de arquivos legítimos. Este arquivo é segmentado em duas partes: i) dados de treinamentos e construção do modelo, que são treinados com os algoritmos de aprendizado de máquina *Support Vector Machine* (SVM), Naive Bayes e árvore de decisão; e ii) dados de teste, que são separados para aferir a desempenho do modelo construído. Ao final, espera-se que a unificação das análises estática e dinâmica seja capaz de classificar novos e atuais tipos de *ransomwares* e determinar qual algoritmo de aprendizado de máquina apresenta melhor desempenho.

Palavras chaves: *ransomware*, análise estática, análise dinâmica, SVM, Naive Bayes, árvore de decisão.

ABSTRACT

Application of machine learning algorithms in the unification of static and dynamic analysis techniques for ransomware classification

Fighting ransomware, a program that blocks and / or encrypts the victim's device for the purpose of requesting a rescue from the encrypted files or the blocked screen, is a difficult task in the face of the rapid dissemination and changes that developers apply to the techniques of dazzle and evasion of programs anti-malware. One of the incentives to the development of this type of malware is that some of the victims choose to pay for the rescue of the information due to the absence of information security practices. This work proposes the unification of two methods used in the analysis and detection of malware, specifically approaching ransomware: static and dynamic analysis. The proposal is to use the integration of techniques to create a file that contains both the characteristics of a ransomware and the characteristics of legitimate files. This file is segmented in two parts: i) training data and model construction, which are trained with the algorithms of machine learning Support Vector Machine (SVM), Naive Bayes and decision tree; and ii) test data, which are separate to gauge the performance of the built model. Finally, it is expected that the unification of the static and dynamic analyzes will be able to classify new and current types of ransomwares and determine which machine learning algorithm performs better.

Keywords: ransomware, static analysis, dynamic analysis, SVM, Naive Bayes, decision tree.

Lista de Ilustrações

Lista de Quadros

Lista de Tabelas

Lista de Abreviaturas e Siglas

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Persistent Threat
BR	Brasil
CBC	Cipher Block Chaining
CD	Compact Disk
CERT	Centro de Estudos para Respostas e tratamento de Incidentes em
CFB	Cipher FeedBack
CRC	Cyclic Redundancy Check
DLL	Dynamic Link Library
FN	Falso Negativo
FP	Falso Positivo
GCC	GNU Compliner Collection
GPS	Global Positioning System
ID	Interative Dichotomizer
IDA	Interactive Disassembler
IoT	Internet of Thing
KVM	Kernel Virtual Machine
MAEC	Malware Attribute Enumeration and Characterization
MD	Message-Digest
MLP	Multi-Layer erceptron
NB	Naive Bayes
NSA	Naional Security Agency
OFB	Output FeedBack
OPCODE	Operation Code
PCAP	packet capture
PDF	Portable Document Format
PE	Portable Executable
RAM	Random Access Memory
RBC	Raciocínio Baseados em Casos
RC	Rivest Cipher
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
RNA	Redes Neurais Artificial
ROC	Receiving Operating Characteristics

Lista de Abreviaturas e Siglas

RSA	Rivest, Shamir e Adleman
SCADA	Supervisory Control and Data Acquisition
SHA	Secure Hash Algorithm
SMB	Server Message Block
SP	*Service Pack*
SSD	Solid State Drive
SVM	Support Vector Machine
TAE	Teoria de Aprendizado Estatístico
TDIDT	Top Down Induction of Decision Trees
TDS	Traffic Distribution System
TFN	Taxa Falso Negativo
TFP	Taxa Falso Positivo
TI	Tecnologia da Informação
TVN	Taxa Verdadeiro Negativo
TVP	Taxa Verdadeiro Positivo
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

Lista de Símbolos

C	:	classe
T	:	tupla
x	:	atributo ou característica de uma classe
x_i	:	vetor de atributos de y_i do registro i
y	:	classe
y_i	:	classe relacionada ao vetor de atributos de i
i	:	registro de um conjunto específico
n	:	quantidade de amostras
m	:	quantidade de atributos
h	:	hipótese ou classificação
f	:	função
p	:	proporção
r	:	subconjunto
κ	:	Kappa
$Pr(a)$	:	concordância dos classificadores
$Pr(e)$	:	probabilidade de concordância da hipótese ao acaso

Sumário

1 INTRODUÇÃO

Com o crescimento da Internet e dos meios de comunicação ocorre o aumento do desenvolvimento de *scripts* e programas maliciosos (*malwares*), como vírus, *ransomwares* e cavalos de Troia. Um dos problemas decorrentes desse aumento é que o desenvolvimento dos métodos de combate não acompanham na mesma proporção a rápida disseminação e propagação destes *malwares*.

1.1 Motivação

Um dos desafios encontrados por especialistas em segurança de Tecnologia da Informação (TI) é o controle do programa malicioso (*malware)* denominado *ransomware,* conforme estudo publicado por Gorman e Mcdonald (2012).

Ransomware é um *malware* que bloqueia o dispositivo e/ou criptografa os arquivos da vítima com o objetivo de solicitar resgate pelos dados bloqueados e/ou criptografados Centro de Estudos para Resposta e Tratamento de Incidentes em Computadores (CERT.BR, 2017). Quando a vítima paga o resgate (geralmente utiliza-se alguma moeda virtual), ela deveria receber a senha para desbloquear o dispositivo ou o código para descriptografar os arquivos. Ressalta-se que o pagamento do resgate não garante o desbloqueio de tela ou o acesso ao código de decriptografia, já que não existe garantia de recebê-los.

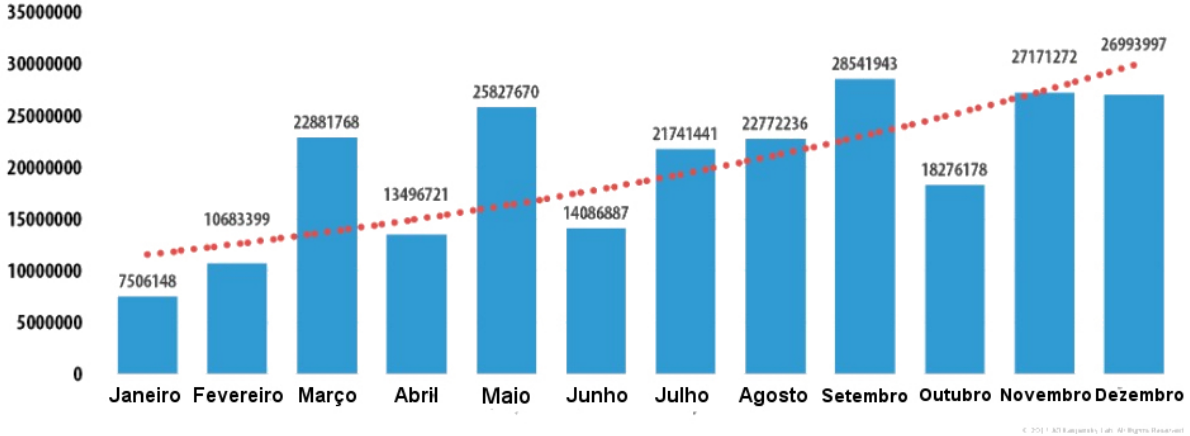
Segundo o CERT.BR (2017), existem duas classes principais de *ransomware:* (1) os *lockers* (travadores); e (2) os *cryptos* (que criptografam). No caso dos *lockers, o ransomware* atua no dispositivo com a finalidade de bloquear o dispositivo da vítima, travando e impedindo o acesso aos arquivos e pastas. Já os *cryptos*, atuam em segundo plano criptografando os arquivos da vítima e exibindo uma mensagem de pedido de resgate.

Existem casos de *ransomwares* que selecionam os arquivos a serem criptografados (.doc., docx, .sql, .dmg, ou .jpg, por exemplo). Com base na seleção do tipo de arquivo a ser criptografado o valor do resgate pode aumentar.

Os meios de contágios mais comuns são: (1) ***e-mail spam*** (utilizando engenharia social); *e (2)* **explorando vulnerabilidades** (CERT.BR, 2017), sendo a engenharia social a tática de contágio principal, em ambos os casos. O autor do *ransomware* faz pressão para que a vítima acesse algum *site* malicioso, ou envia por *e-mail* anexos infectados. De acordo com o estudo da Kaspersky (2017), 20% dos *e-mail* com conteúdo considerado

spam em 2016 propagam *ransomwares*, a Figura 1 apresenta a quantidade de *e-mails* considerados *spam* em 2016.

Figura 1: Quantidade de *e-mail spam* maliciosos em 2016.

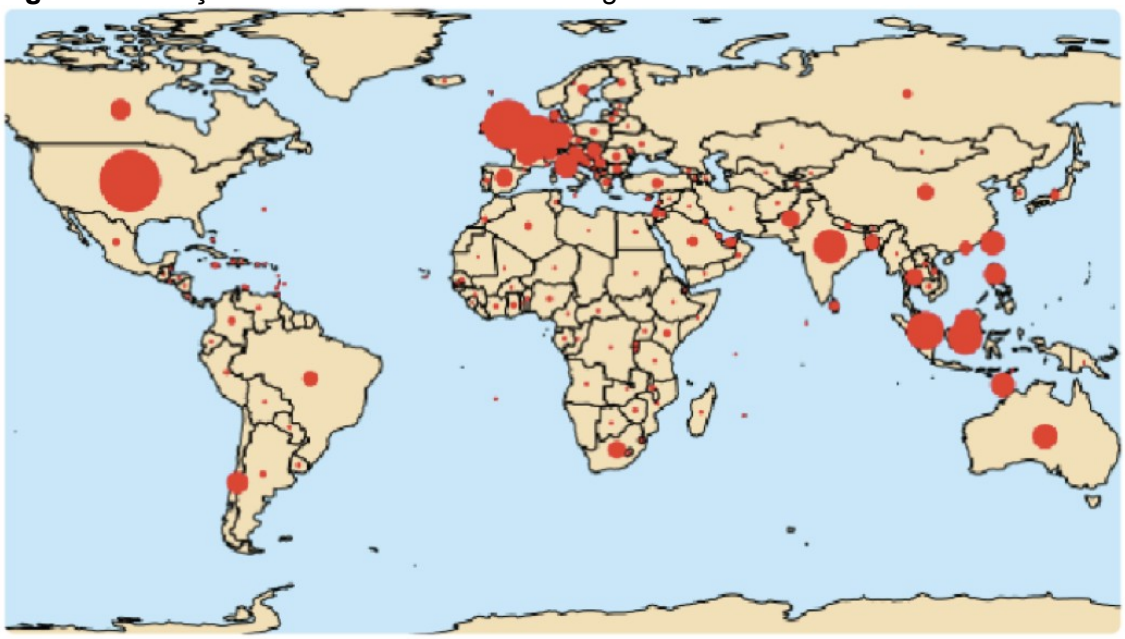


Fonte: Kaspersky Lab, (2017).

A Figura 1 mostra no gráfico a evolução mês a mês da quantidade de *e-mails* considerados como *spam* durante o ano de 2016.

Sophos (2017) realiza um estudo com o objetivo de verificar quais são os *ransomwares* mais ativos, ou seja, que infectaram o maior número de dispositivos e quais países são propensos à infecção em escala mundial (Figura 2).

Figura 2: Infecção de *ransomware* em escala global.



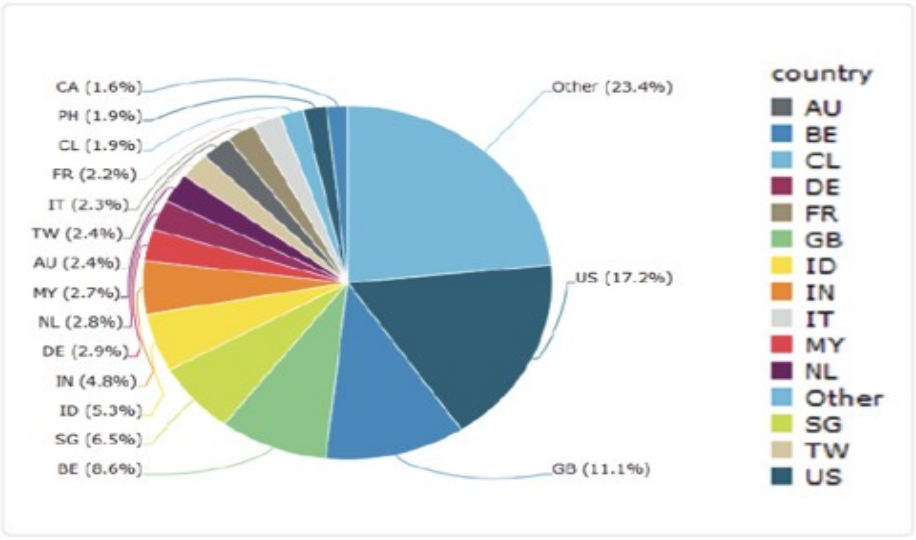
Fonte: Kaspersky, 2017

Conforme pode ser visto na Figura 2, observa-se que *ransomware* possui alcance global e que poucos países, como por exemplo a Namíbia, o Sudão do Sul e a Libéria,

podem não ter dispositivos contaminados. Ainda, percebe-se que países desenvolvidos tecnologicamente como os Estados Unidos da América e o Japão estão propensos a ter maior impacto com os danos causados pelos *ransomwares* do que países com baixo desenvolvimento tecnológico, como o Peru e Equador.

O estudo de Sophos (2017) apresenta informações detalhadas sobre países infectados e os *ransomwares* mais ativos, conforme pode ser visto nas Figuras 3 e 4.

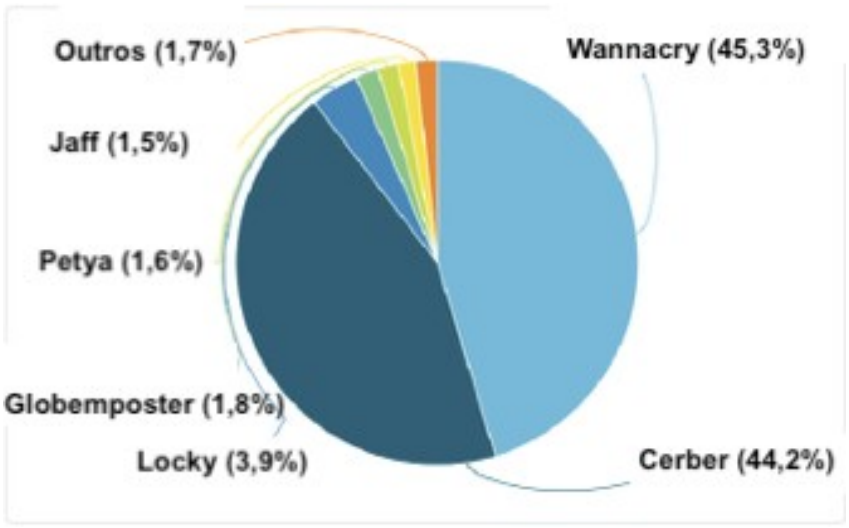
Figura 3: Países infectados e nível de infecção.



Fonte: Sophos, 2017

Como pode ser visto na Figura 3, os países em que os *ransomwares* são mais ativos, ou seja, com maior número de infecções são os Estados Unidos da América (17,20%), o Reino Unido (11,1%) e a Bélgica (8,6%).

Figura 4: Principais ransomwares ativos.



Fonte: Adaptado de Sophos, 2017

A Figura 4, apresenta os *ransomwares* que mais infectaram dispositivos e como pode ser observado o Cerber e as variantes do Wannacry são responsáveis por 89,5% das infecções realizadas durante o estudo.

O método de propagação dos *ransomwares* é muito rápido comparado à evolução dos métodos de combate a este tipo de *malware*. Nos trabalhos de Andronio (2012), Hosfelt (2015), Caldas (2016), Schmidt *et. al.* (2014) e Sgandurra *et. al.* (2016) observam-se estudos que apontam três direções para a detecção e classificação de *ransomware*: (1) técnica de análise estática; (2) técnica de análise dinâmica; e (3) aprendizado de máquina.

Segundo Sikorski (2012), a técnica de análise estática (estrutural) é o processo de análise do código ou da estrutura de um programa para determinar a sua função, sendo, em geral, realizada sem que o arquivo seja executado no sistema. Um exemplo de técnica de análise estática é o uso de engenharia reversa, cujo objetivo é a extração do código fonte para análise. Este processo é demasiadamente demorado, principalmente se for considerada a velocidade com que os *ransomwares* são desenvolvidos e disseminados.

A técnica de análise dinâmica (comportamental), requer um ambiente seguro, controlado e confinado onde o arquivo considerado suspeito seja executado sem causar qualquer tipo de dano a máquina hospedeira (SIKORSKI, 2012). O arquivo suspeito é enviado para esse ambiente virtual e os comandos e instruções são registrados com a finalidade de classificar o arquivo suspeito como maligno ou benigno. Um exemplo desse tipo de análise é o uso de máquina virtual, em conjunto com o programa Cukoo SandBox (https://cuckoosandbox.org).

Já o uso de aprendizado de máquina, permite ao sistema adquirir conhecimento de forma automática e tomar decisões com base no treinamento realizado e no algoritmo utilizado (MITCHELL, 1997 e CARVALHO, 2011). A Seção 2, Teoria, contextualiza o assunto.

Detectar *ransomwares* não é uma tarefa fácil pois os desenvolvedores usam novas e sofisticadas técnicas de ofuscamento, análise e evasão dos programas de antivírus (SGANDURRA *et. al.*, 2016).

Segundo Malin *et. al.* (2012), cinco fases são utilizadas para identificar *malwares*: (1) preservação forense e exame dos dados voláteis: compreende a preservação dos dados voláteis (enquanto o sistema está em operação); (2) exame da memória *Random Access Memory* (RAM) do dispositivo infectado; (3) análise forense, com a verificação dos dispositivos de armazenamento permanente, como disco rígido; (4) verificação das

características do arquivo desconhecido: inicialmente se verifica se o arquivo é de origem conhecida ou desconhecida, em seguida é realizada uma pré-análise estática que verifica o cabeçalho do arquivo com a finalidade de tentar classificá-lo como benigno ou maligno; e (5) são usadas ferramentas baseadas nas técnicas de análises estática e/ou dinâmica no arquivo considerado suspeito.

Schultz *et. al.* (2001) são os primeiros a introduzir o conceito de aplicação de algoritmos de aprendizado automático para a detecção de *malware*, com base nos respectivos códigos binários, aplicando diferentes classificadores como *Repeated Incremental Pruning to Produce Error Reduction* (RIPPER), Naive Bayes e a combinação de classificadores considerando três tipos de conjuntos de atributos: (1) cabeçalhos de programas; (2) cadeias; e (3) sequências de *bytes*.

Há uma série de trabalhos que utilizam as técnicas de detecção de *ransomwares* descritas – Sgandurra *et. al.* (2016) usa o aprendizado de máquina, Andronio (2012) e Nath e Mehtre (2014 usam análise estática. Hosfelt (2015) usa análise estática em conjunto com aprendizado de máquina e, por fim Caldas (2016) e Schmidt *et. al.* (2014) usam análise dinâmica. Apesar dos resultados obtidos, todos estes trabalhos possuem limitações e vulnerabilidades que impedem que a análise seja realizada por completo. Contudo, pode ser observado que são raríssimos os trabalhos que combinam a utilização das técnicas de análise estática e dinâmica e implementem algoritmos de aprendizagem de máquina para classificação de *ransomwares*.

1.2 Objetivo

A presente dissertação apresenta como objetivo geral criar um vetor de aprendizagem de máquina que permita o agrupamento das técnicas de análise estática e dinâmica para identificação e classificação de *ransomwares.*

Os objetivos específicos são:

- Comparar os algoritmos de aprendizagem de máquina utilizados: *Support Vector Machine* (SVM), Naive Bayes e árvore de decisão.
- Unificar os relatórios das técnicas para criação do vetor de aprendizagem de máquina;
- Tratar o vetor para ser submetido aos algoritmos de aprendizagem de máquina.

1.3 Contribuições

Criar um método para unificar as técnicas de análises estática e dinâmica, resultando em um vetor que possa ser utilizado para classificação de *ransomwares* com o uso de algoritmo de aprendizado de máquina.

Comparar a precisão, a revocação e o desempenho dos algoritmos de classificação por meio da concordância Kappa e de métodos baseados na matriz de confusão.

1.4 Método de trabalho

A presente subseção descreve as etapas e atividades que são desenvolvidas para alcançar o objetivo da dissertação.

O método de trabalho da presente dissertação é dividido em 3 etapas:

1. Revisão Bibliográfica: o presente trabalho tem início com o levantamento bibliográfico sobre os assuntos relacionados a aprendizado de máquina supervisionado, análise estática e dinâmica. O objetivo é estudar os meios utilizados para identificação de *ransomware*:

2. Experimento: um ambiente é montado e um experimento é realizado para a comprovação das hipóteses levantadas. Nesta etapa são realizadas as seguintes atividades:

1.1 Preparação do Ambiente: nesta etapa o ambiente é preparado com a instalação e configuração dos programas necessários;

1.2 Obtenção das Amostras: são coletadas as amostras limpas, ou seja, sem qualquer tipo de *malware,* e as amostras contaminadas com *ransomwares* para compor o conjunto do aprendizado;

1.3 Escolha dos atributos: nesta etapa são selecionados quais atributos devem ser avaliados;

1.4 Análises: as amostras são submetidas às análises estática e dinâmica com o objetivo de gerar um relatório com os valores dos atributos;

1.5 Vetor de aprendizagem de máquina: nesta etapa são realizadas a unificação das análises estáticas e dinâmicas para gerar o vetor de aprendizagem de máquina;

1.6 Aprendizagem de máquina, validação e avaliação: o vetor de aprendizado de máquina é utilizado pelos algoritmos de aprendizagem e submetidos ao método de validação para que possa ser avaliado;

1.7 Coleta dos dados e análise de resultados: são colhidos os resultados para análise. O objetivo desta seção é aferir se o objetivo é alcançado.

1.5 Organização do trabalho

A presente dissertação é organizada em seis seções, além desta seção de introdução:

Na Seção 2, Teoria, apresentam-se os conceitos sobre aprendizagem de máquina, algoritmos de aprendizagem, *frameworks* de aprendizagem, validação e avaliação. Contextualiza *ransomwares, a*presenta conceitos sobre as análises estática e dinâmica, técnicas de ofuscamento e evasão.

Na Seção 3, Trabalhos Relacionados, são apresentadas pesquisas sobre o combate e explanação dos temas como *ransomware,* aprendizado de máquina e as análises estática e dinâmica.

A Seção 4, Proposta de Unificação e Aprendizado de Máquina, descreve como é realizado o processo de unificação das análises estática e dinâmica para posterior processo de classificação do *ransomware* pelo aprendizado de máquina supervisionado. Nesta seção são apresentados em detalhes os processos de obtenção, seleção e predição das amostras para formar um conjunto com exemplos para ser submetido aos algoritmos de aprendizagem de máquina

A Seção 5, Coleta e Análise de Resultados, os resultados são coletados e são comparados os métodos de avaliação kappa, precisão, acurácia e revocação para aferir qual algoritmo obtém o melhor desempenho.

Por fim, a Seção 6, Considerações Finais, retoma o objetivo do trabalho e faz um resumo e uma análise dos resultados obtidos. Também são elencadas as contribuições reais e os trabalhos futuros que podem ser realizados a partir desta dissertação.

2 TEORIA

Nesta seção são apresentados os fundamentos, contextualização e conceitos essenciais para a compreensão deste trabalho. São elencados os conceitos e uma breve contextualização sobre aprendizado de máquina, algoritmos, principais métricas utilizadas para verificar a qualidade e o desempenho das hipóteses, ataques computacionais, *ransomware* e os vetores de contágio. Por fim, são apresentados os conceitos de análise estáticas e dinâmica e os métodos de ofuscamento e evasão.

2.1 Aprendizado de Máquina

Segundo Mitchell (1997), o aprendizado de máquina é uma área da inteligência artificial que permite processar e adquirir conhecimento de forma automática. No aprendizado de máquina utiliza-se a indução como forma de inferência lógica para obter conclusões genéricas sobre um conjunto de amostras particular (CARVALHO, 2011).

Monard e Baranauskas (2003) dizem que a indução é um processo de generalização pela observação e combinação de amostras particulares. Na indução, um conceito é aprendido efetuando-se a inferência indutiva sobre as amostras apresentadas.

Indutor é o tipo de algoritmo (ou algoritmo de indução) de aprendizado de máquina utilizado a partir de um conjunto de amostras. O objetivo de um indutor é extrair um bom classificador a partir de um conjunto de amostras devidamente conhecidas (MONARD; BARANAUSKAS, 2003).

O aprendizado indutivo possui três métodos para aprendizagem de máquina. No **aprendizado supervisionado**, segundo Monard e Baranauskas (2003), o indutor gera as hipóteses das amostras de um conjunto. O indutor recebe um conjunto de amostras de treinamento constituído por várias instâncias que possuem atributos e rótulos de classe e que ao final do processo de indução recebem uma predição de classe. Utilizando um conceito mais simples, são apresentadas ao computador amostras de entradas e saídas desejadas que são fornecidas por um professor. O objetivo é aprender uma regra que mapeie as entradas para as saídas.

O aprendizado de máquina supervisionado pode ser de dois tipos: (1) **classificação**, quando os valores de classe são nominais (discretos), como por exemplo: rotular o arquivo em *ransomware* ou benigno; e (2) **regressão** quando os valores são reais (contínuos), por exemplo: calcular a previsão de produtos a serem vendidos durante um evento (MONARD; BARANAUSKAS, 2003). Conceitos devem ser apresentados para o entendimento do aprendizado de máquina supervisionado:

- **Classe (** C **)** é o rótulo (nome) que descreve o fenômeno de interesse. Exemplo: um determinado arquivo de sistema pode ser *ransomware* ou não *ransomware (MONARD; BARANAUSKAS, 2003);*
- **Amostra**, também conhecido como dado, caso, exemplo ou registro é uma tupla T de valores de **atributos (características).** Amostras são tuplas $T_i=(x_{i1}, x_{i2}....., x_{\beth}, y_i)=(\vec{x}_i)$, também denotado por (x_i, y_i) , onde (x_i) é um vetor de atributos e y_i é uma classe, tal que $y_i \in \{C_i, C_2, ..., C_n\}$ (MONARD; BARANAUSKAS, 2003).

No **aprendizado não supervisionado**, o indutor agrupa as amostras em grupos que posteriormente são analisados para identificação dos significados dos grupos (MONARD; BARANAUSKAS, 2003). Um conceito mais simples é utilizado por Luger (2008) que afirma que o **aprendizado não supervisionado** elimina o professor e requer que o próprio algoritmo de aprendizado avalie os conceitos. A ciência talvez seja o melhor exemplo de aprendizado não supervisionado em seres humanos.

Já o **aprendizado por reforço** é muito utilizado pela robótica. Onde a observação do ambiente é utilizado para adquirir o aprendizado (LUGER, 2008).

2.1.1 Algoritmos de aprendizado de máquina

O aprendizado de máquina baseia-se em vários padrões estruturais (MONARD; BARANAUSKAS, 2003) e os algoritmos de aprendizado de máquina são responsáveis por organizar padrões em um conjunto de dados.

Monard e Baranauskas (2013) descrevem estes padrões (sistemas de aprendizagem) em:

- **Simbólicos:** buscam aprender construindo representações simbólicas de conceito através da análise de exemplos e contra-exemplos desse conceito. As representações simbólicas estão tipicamente na forma de alguma expressão lógica, árvore de decisão, regras ou semântica;
- **Estatísticos:** a ideia geral consiste em utilizar modelos estatísticos para encontrar uma boa aproximação do conceito induzido. O algoritmo Bayesiano se destaca como método estatístico;
- **Baseado em Exemplos:** uma forma de classificar um exemplo é lembrar de outro similar cuja classe seja conhecida e assume que o novo exemplo terá a mesma

classe. Os algoritmos de *Nearest Neighbours* e Raciocínio Baseados em Casos (RBC) são, provavelmente, técnicas mais conhecidas;

- **Conexionista:** baseado na estrutura do modelo biológico do sistema nervoso. O algoritmo de Redes Neurais Artificiais (RNAs) é um exemplo desse tipo de sistema de aprendizado;
- **Evolutivo:** baseado no modelo biológico de aprendizado. Esse tipo de sistema de aprendizado possui uma analogia direta com a Teoria de Darwin de seleção natural. Elementos que possuem desempenho fraco são descartados, enquanto os elementos mais fortes proliferam, produzindo variações de si mesmos. Exemplo: algoritmos genéticos.

Os algoritmos de ***Support Vector Machines*** **(SVM)** são um algoritmo de aprendizado de máquina supervisionado embasados pela Teoria de Aprendizado Estatístico (TAE), que estabelece uma série de princípios que devem ser seguidos na obtenção de classificadores com boa capacidade de generalização (CARVALHO, 2011).

Carvalho *et. al.* (2011) conceitua os algoritmos de **Árvore de Decisão** como a estratégia de dividir para conquistar para resolver o problema de decisão. Um problema complexo é dividido em problemas mais simples, aos quais recursivamente é aplicada a mesma estratégia. As soluções dos subproblemas podem ser combinadas, na forma de uma árvore, para produzir uma solução do problema complexo.

Outra forma de realizar o processo de predição é o uso do algoritmo de aprendizagem de máquina **Naive Bayes (NB)** ou **bayesiano**. Os **algoritmos bayesianos** são baseados no teorema de Bayes (métodos probabilísticos bayesianos) que são utilizados quando as informações disponíveis são incompletas ou imprecisas.

Os métodos probabilísticos bayesianos assumem que a probabilidade de um evento A, que pode ser uma classe (por exemplo, um doente apresentar uma determinada doença), dado um evento B, que pode ser um valor para um atributo de entrada (por exemplo, ter resultado positivo em um exame de raios-X), não depende apenas da relação entre A e B, mas também da probabilidade de observar A independentemente de observar B (MITCHELL, 1997).

2.1.2 Validação e avaliação em aprendizado de máquina

Na aplicação de algoritmos de aprendizado de máquina há problemas reais, em geral, o conhecimento que se tem do domínio sendo investigado é provido unicamente pelo conjunto de exemplos, a partir do qual a indução de um modelo preditivo/descritivo é

então realizada. Em geral, pode-se afirmar que não existe técnica universal, ou seja, não é possível estabelecer a priori que uma técnica de aprendizado de máquina em particular se sairá melhor na resolução de qualquer tipo de problema (CARVALHO *et. al.*, 2011).

> "[…] as próprias características das técnicas existentes e do problema que está sendo solucionado podem ser consideradas para auxiliar na escolha da técnica a ser utilizada sobre um novo conjunto de dados. Em domínios em que os exemplos possuem alta dimensionalidade, as SVMs são boas candidatas enquanto o algoritmo *k*-NN usando a distância euclidiana pode, a princípio, não parecer uma escolha adequada. Caso seja necessário que o modelo obtido seja interpretável, técnicas simbólicas, como árvores de decisão, podem ser preferíveis a modelos caixa-preta como os gerados pelas Redes Neurais Artificiais e pelas SVMs." (CARVALHO *et. al.,* 2011).

Assim, conclui-se que, qualquer algoritmo pode ser candidato à solução de um dado problema o que evidencia a necessidade de realizar experimentos que precisem ser validados e avaliados. Ainda assim, o entendimento de alguns conceitos para a compreensão dos processos de avaliação e validação são necessários (MONARD; BARANAUSKAS, 2003; CARVALHO, 2011):

- ***Overfitting*:** ocorre quando uma hipótese apresenta uma baixa capacidade de generalização, a razão pode ser que ela está superajustada aos dados de treinamento. Também é dito que a hipótese memorizou ou se especializou nos dados de treinamento;
- ***Underfitting*:** neste caso, a hipótese apresenta uma baixa taxa de acerto mesmo no subconjunto de treinamento, configurando uma condição de subajustamento. Essa situação pode ocorrer, por exemplo, quando os exemplos de treinamentos disponíveis são pouco representativos ou o modelo usado é muito simples e não captura os padrões existentes nos dados;
- **Ruído:** os valores dos atributos e rótulos de classe podem possuir imperfeições. Estas imperfeições são chamadas de **ruídos**;
- **Poda:** utilizado em algoritmos de árvore de decisão. É considerada a parte mais importante do processo de construção da árvore, pelo menos em domínios com ruídos. Dados ruidosos levantam dois problemas: (1) as árvores induzidas classificam novos objetos em um modo não confiável. Nós mais profundos refletem mais o conjunto de treinamento (*overfitting)* e aumentam o erro devido à variância do classificador; e (2) a árvore induzida tende a ser grande o que dificulta a compreensão. Métodos de poda podem ser divididos em dois grupos principais: **Pré-poda:** ocorre durante o processo de construção do modelo indutor. Conta com regras de parada que previnem a construção daqueles ramos que não parecem melhorar a precisão preditiva da árvore e possui a vantagem de não perder tempo construindo uma estrutura que não é usada na árvore

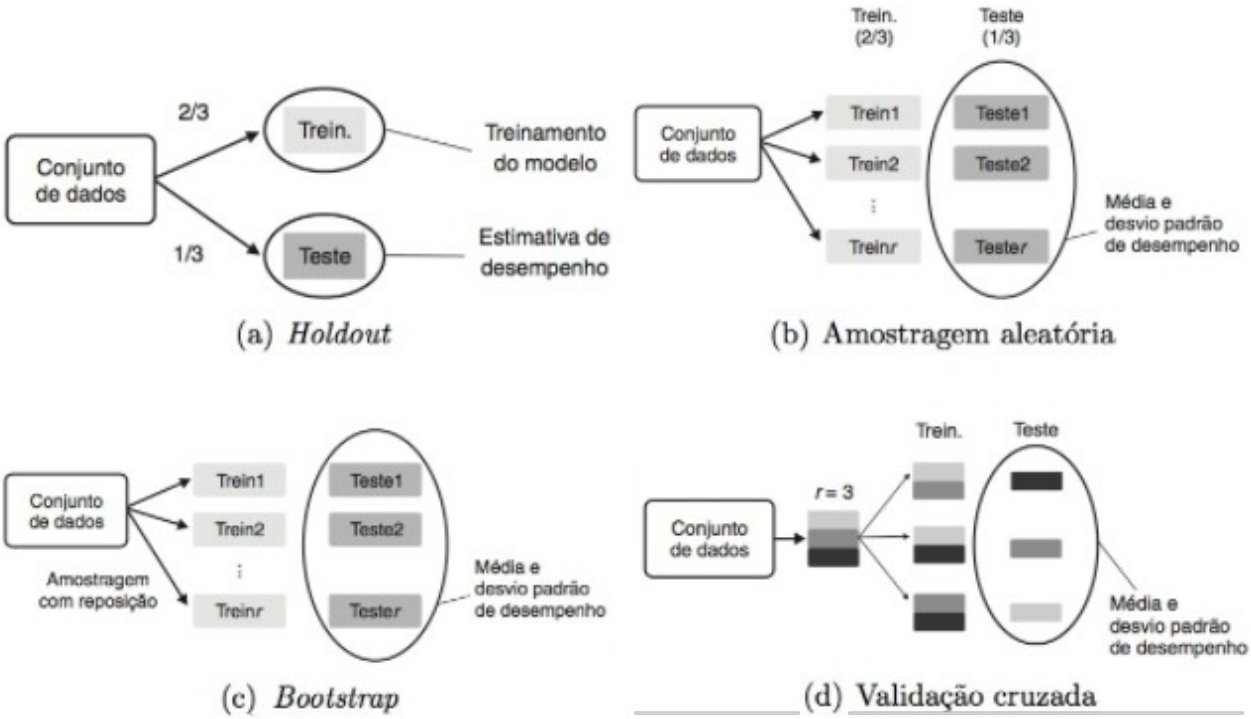
final; e **Pós-poda:** ocorre após a construção do modelo indutor. Uma árvore completa e superajustada (*overfitting*) aos dados de treinamento, é gerada e podada posteriormente;

- **Bias ou viés indutivo:** é a diferença entre o valor do parâmetro e o valor produzido pelo indutor. Quando um algoritmo de aprendizado de máquina está aprendendo a partir de um conjunto de dados de treinamento, ele está procurando uma hipótese, no espaço de possíveis hipóteses, capaz de descrever as relações entre os objetos e que melhor se ajustem aos dados de treinamento;
- **Completude e consistência:** Uma hipótese pode ser avaliada a respeito de: (a) sua completude, se ela classifica todos os exemplos; e (b) sua consistência, se ela classifica corretamente todos os exemplos;
- **Acurácia:** é o número de Verdadeiros Positivos (VP) e Verdadeiros Negativos (VN) dividido pelo número total de exemplos;
- **Taxa de erros:** é o complemento da acurácia ou soma dos Falsos Positivos (FP) e Falsos Negativos (FN) dividido pelo número total de exemplos.

Segundo Carvalho *et. al.* (2011), a validação de qualquer nova técnica de aprendizado de máquina proposta geralmente envolve a realização de experimentos controlados, em que se demonstre a sua efetividade na solução de diferentes problemas, representados por seus conjuntos de dados associados. Dessa forma, é recomendável seguir procedimentos que garantam a corretude (ou seja, para cada entrada ele produz a saída correta), a validade e a reprodutividade dos experimentos realizados e, mais importante, das conclusões obtidas a partir de seus resultados. A Figura 5 apresenta alguns dos principais métodos de validação existentes, que são brevemente apresentados.

No caso do ***holdout***, divide-se o conjunto de dados em uma proporção de p para treinamento e $1-p$ para teste, conforme Figura 5 (a). Normalmente, emprega-se $p=\frac{2}{3}$. Esse tipo de separação pode subestimar a taxa de acerto, uma vez que um preditor produzido sobre todos os objetos em geral apresenta uma taxa de acerto maior que a gerada a partir de uma parte deles (MONARD; BARANAUSKAS, 2003). No entanto, segundo Carvalho (2011), para conjuntos de dados grandes isso não representa um problema.

Figura 5: Principais métodos de validação.



Fonte: Carvalho *et. al.* (2011).

Já na **amostragem aleatória**, são realizadas diversas partições aleatórias para obter uma média de desempenho em *holdout*. Assim, os resultados são menos dependentes da partição feita. A Figura 5 (b) apresenta um exemplo em que são gerados r diferentes subconjuntos aleatórios.

No método ***bootstrap***, r subconjuntos de treinamento são gerados a partir do conjunto de exemplo original, conforme Figura 5 (c). Os exemplos são amostrados aleatoriamente desse conjunto, com reposição. Logo, um exemplo pode estar presente em um determinado subconjunto de treinamento mais de uma vez, ou seja, o processo de classificação de repete diversas vezes. Os conjuntos não selecionados compõem os subconjuntos de teste. O resultado final é dado pela média do desempenho observado em cada subconjunto de teste.

No método de **validação cruzada (*cross-validation) r-fold***, segundo Carvalho et. al. (2011), o conjunto de exemplos é dividido em r subconjuntos de tamanho aproximadamente iguais. Os objetos de $r-1$ partições são utilizados no treinamento de um preditor, o qual é então testado na partição restante. Esse processo é repetido r vezes, utilizando em cada ciclo uma partição diferente para teste. O desempenho final do preditor é dado pela média dos desempenhos observados sobre cada subconjunto de teste. Esse processo é ilustrado na Figura 5 (d), empregando $r=3$.

Diversos são os métodos de avaliação em aprendizado de máquina. No entanto, alguns métodos compartilham notações. Por exemplo, uma classe é denotada positiva (+) enquanto outra denotada negativa (-). Outras notações comuns são (CARVALHO *et. al.*, 2011):

- Verdadeiro Positivo (**VP)**: corresponde ao número de exemplos da classe positiva classificada corretamente;
- Verdadeiro Negativo (**VN)**: corresponde ao número de exemplos da classe negativa classificada corretamente;
- Falso Positivo (**FP)**: corresponde ao número de exemplos cuja a classe verdadeira é negativa mas que foram classificados incorretamente como pertencendo à classe positiva;
- Falso Negativo (**FN)**: corresponde ao número de exemplos originalmente pertencentes à classe positiva que foram incorretamente preditos como da classe negativa;
- n = VP + VN + FP + FN.

Muitos métodos de avaliação se baseiam nas métricas da matriz de confusão. Uma **matriz de confusão** de uma hipótese oferece uma medida efetiva do modelo de classificação, o eixo horizontal determina os valores de classe corretos enquanto o eixo vertical denota os valores de classe atribuídos pelo classificador. A Figura 6 ilustra uma matriz de confusão.

Figura 6: Matriz de confusão para um problema com duas classes.

		Classe predita	
		+	-
Classe verdadeira	+	VP	FN
	-	FP	VN

Fonte: Carvalho *et. al.* (2011)

Conforme pode ser observado, a Figura 6 apresenta uma matriz de confusão de duas classes preditas.

A partir da matriz de confusão, uma série de outas medidas de desempenho podem ser derivadas, como (MONARD; BARANAUSKAS, 2003):

- **Acurácia:** é o número de VP e VN dividido pelo número total de exemplos;

$$Acc = \frac{VP + VN}{VP + FP + VN + FN} \quad (1)$$

- **Precisão:** proporção de exemplos positivos classificados corretamente entre todos aqueles preditos como positivos por $\hat{f}$;

$$prec(\hat{f}) = \frac{VP}{VP + FP} \quad (2)$$

- **Sensibilidade ou revocação:** correspondente à taxa de acerto na classe positiva. Também é chamada de taxa de verdadeiros positivos (TVP);

$$sens(\hat{f}) = rev(\hat{f}) = TVP(\hat{f}) = \frac{VP}{VP + FN} \quad (3)$$

Uma métrica utilizada para indicar a concordância entre dois classificadores é a **Estatística (ou Coeficiente) Kappa -** κ **.** O Coeficiente Kappa é uma medida estatística que leva em consideração as probabilidades de as concordâncias de itens categóricos terem ocorridos ao acaso, conforme a fórmula (COHEN, 1960):

$$\kappa = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \quad (4)$$

- κ : valor resultante do coeficiente Kappa;
- $Pr(a)$: é a proporção de vezes que os classificadores concordam (resultado do aprendizado de máquina). É obtida através de:

$$Pr(a) = Acc \quad \text{ou} \quad Pr(a) = \frac{VP + VN}{VP + VN + FP + FN} \quad (5)$$

- $Pr(e)$: é a proporção de vezes que os κ classificadores esperam concordar ao acaso ou a probabilidade de ocorrer a concordância da hipótese ao acaso. Sua obtenção é realizada através da fórmula:

$$Pr(e) = \left(\left(\frac{VP + FN}{TOTAL} * \frac{VP + FP}{TOTAL} \right) + \left(\frac{VN + FP}{TOTAL} * \frac{VN + FN}{TOTAL} \right) \right) \quad (6)$$

O coeficiente Kappa é uma importante métrica para determinar o quão bem funciona a aplicação de alguma medição. Geralmente varia entre 0 e 1, no entanto números negativos são possíveis como resultado kappa (κ). Assim, conclui-se que: (a) resultados mais próximos de +1, significam uma concordância dos classificadores; (b) resultados próximos a 0, significam que não existem relação ou uma pequena concordância entre os classificadores; e (c) resultados abaixo de 0, os classificadores não concordam, ou seja, classificam exatamente o oposto. Para padronizar o método de avaliação Landis e Kock (1977) desenvolvem o índice de concordância Kappa, apresentado na Tabela 1.

Tabela 1: Índice de concordância Kappa.

K	Interpretação
< 0	Nenhuma concordância
0 a 0,2	Leve concordância
0,21 a 0,40	Concordância regular
0,41 a 0,60	Concordância moderada
0,61 a 0,80	Concordância substancial
0,80 a 1,0	Concordância quase perfeita

Fonte: Adaptado de Landis e Kock (1977).

2.1.3 *Framework* de aprendizado de máquina

Um *framework* é uma abstração que une códigos comuns entre vários projetos de *software* provendo uma determinada funcionalidade genérica. O *framework* pode ser visto como um conjunto de classes que, em conjunto, realizam determinada função para um domínio de um subsistema da aplicação. A estrutura de um *framework* representa mais do que somente um conjunto de classes, pois também define como as instâncias das classes são permitidas a interagir entre si em tempo de execução. Efetivamente, o *framework* age como uma espinha dorsal, no que diz respeito a como as classes estarão relacionadas entre si (RIEHLE, 2000).

Um *framework* é construído utilizando-se de implementações reusáveis na forma de classes concretas e abstratas. Geralmente, o *framework* é desenvolvido utilizando bibliotecas e integrando *Application Programming Interface* (API). API é um conjunto de rotinas e padrões estabelecidos por um determinado programa para utilização das suas funcionalidades por outros programas ou aplicativos que não desejam desenvolver aqueles recursos que já foram desenvolvidos anteriormente por outra empresa ou programador, o objetivo é usar apenas o serviço.

Scikit-learn (http://scikit-learn.org/) é um *framework* de aprendizado de máquina de código aberto para a linguagem de programação Python. O objetivo do desenvolvimento do scikit-learn é oferecer soluções simples e eficientes aos problemas de aprendizado de máquina (PEDREGOSA, 2011). O Scikit-learn disponibiliza diversos módulos, como por exemplo (SCIKIT-LEARN, 2017):

- Aprendizado Supervisionado: Regressão Linear, Logística e Bayesiana, Gradiente Descendente, *Support Vector Machine* – SVM e Naive Bayes;
- Aprendizado Não-Supervisionado: *K-Means*, *Affinity Propagation*, *Principal Component Analysis (*PCA) e *Independent Component Analysis (*ICA);

- Transformação de base de dados: pré-processamento de dados, extração de características em texto e imagens e aproximação de kernel;
- Validação Cruzada: Permutação Aleatória, *Leave-One-Out* e *Leave-P-Out*;
- Base de dados de exemplos e possibilidade de *download* de certas bases disponíveis na *internet*.

O Python é uma linguagem de programação que tem como objetivo tornar o trabalho de desenvolvimento mais rápido e integrado a diferentes sistemas. O estilo que sua sintaxe apresenta é próximo a um pseudo-código, o que torna a leitura do código intuitiva e de fácil interpretação (SUMMERFIELD, 2009).

2.2 *Ransomware*

Ransomware é um tipo de código malicioso que torna inacessível os dados armazenados em um equipamento, geralmente usando criptografia, e que exige pagamento de resgate para restabelecer o acesso ao usuário (CERT.BR, 2017). Quando a vítima paga o resgate, geralmente utilizando moeda virtual, recebe a senha de desbloqueio ou código para decriptografar seus arquivos.

A motivação financeira aparece como um dos principais motivadores para desenvolvimento de *ransomwares* e o meio de pagamento, geralmente, é a moeda virtual (também conhecida com moeda digital ou criptomoeda) como, por exemplo, *bitcoin* e *ethereum* o que dificulta o rastreamento dos valores pagos.

A moeda virtual não possui um país ou Banco Central regulador, o responsável por sua “criação” é um programa de computador conhecido como minerador. No caso específico da moeda virtual Bitcoin, a negociação é realizada através de um par de chaves criptográficas. A negociação se junta ao conjunto de blocos, denominado cadeia de blocos (*blockchain*) onde é criado um *hash*. Assim é inviável e até então impossível rastrear os valores pagos.

Um *ransomware* possui características que o distingue dos demais *malwares*. Possui um **texto de ameaça –** em que solicita o resgate para que o usuário tenha seus arquivos novamente (ANDRONIO, 2012), **acessa uma grande quantidade de arquivos em um curto período de tempo** (CALDAS, 2016), **cria arquivos criptografados** (CALDAS, 2016) e **bloqueia o dispositivo** – impedindo o usuário de manipular seus arquivos (ANDRONIO, 2012).

Em Maio/2017, o *ransomware* WannaCry 2.0 explorou uma vulnerabilidade no protocolo de compartilhamento de arquivo e pastas, o *Server Message Block* (SMB). Apesar da Microsoft já ter disponibilizado em 14 de março de 2017 (para Windows *Desktop*, Vista, 7, 8 e 10 e para Windows Server 20018, 2012 e 2016) uma correção para essa vulnerabilidade (*patch* de segurança MS17-010), a disseminação do WannaCry 2.0 foi rápida e impactante. A rápida disseminação deve-se ao fato de grande parte dos computadores com os sistemas operacionais Microsoft estarem desatualizados e, ainda, acredita-se que foi graças a um *exploit,* chamado ExternalBlue, criado pela Agência de Segurança Nacional Americana – *National Security Agency* (NSA) – ter sido publicado na internet pelo grupo *hacker Shadow Brokers.*

Os usuários do sistema operacional para *desktop* da Apple, Mac OS X, em fevereiro de 2017 foram vítimas de ataques de *ransomware patcher.* O *patcher* usa a estratégia de se passar por um *crack* (termo utilizado para programas que tem o objetivo de fraudar a licença de um programa pago, obtendo assim o direito de uso sem a necessidade de adquirir/compra a licença devida do programa em questão) dos programas como Adobe Premiere Pro e Microsoft Office. Após a infecção o *ransomware patcher* solicita o valor de 0,25 bitcoins para a chave de decriptografia dos arquivos. Neste caso, mesmo que o valor fosse pago a chave de decriptografia nunca era enviada e os arquivos não poderiam ser acessados.

Em Julho de 2017, os usuários do sistema operacional Android foram atacados pelo *ransomware* LeakerLocker que estava incorporado no pacote de instalação dos programas Wallpapers Blur HD e o Booster & Cleaner Pro que poderiam ser instalados através da loja de aplicativos da Google, o Google Play, gratuitamente. Após a infecção o LeakerLocker bloqueia a tela do dispositivo com um texto ameaçador informando que divulgará o conteúdo do celular (videos, fotos, mensagens de texto, histórico de ligações, mensagens do Facebook, histórico de navegação e histórico de localização via *Global Positioning System* – GPS) em 72 horas caso não seja pago o valor de U$$ 50,00 (cinquenta dólares). Estima-se que os dois programas tenham sido baixados pelo menos 15 mil vezes desde abril de 2017, antes de serem removidos pelo Google.

Sgandurra *et. al.* (2016) descreve as principais técnicas de disseminação do *ransomware*, que são: *e-mail spam, phishing, Exploit kits, Downloader, Trojan Botnets*, táticas de engenharia social e sistemas de distribuição de tráfego, mais conhecido como *Systems Distribution Traffic* (TDS).

As recomendações para precauções com *ransomware* (e para *malwares* em geral) são (CERT.BR, 2017; SGANDURRA *et. al.*, 2016):

- Sempre ter um antivírus instalado e atualizado;
- Realizar *backup* constante das informações e, preferencialmente, fora do ambiente local;
- Nunca abrir *link* e e-mail de origem desconhecida;
- Desconfiar de *e-mails* de lojas *online*, bancos, polícia ou receita federal;
- Não realizar *downloads* de programas que não sejam de fontes confiáveis;
- Em um ambiente corporativo, realizar treinamentos constantes.

Ter uma **rotina de *backup*** ainda é a única forma de se recuperar rapidamente de um ataque de *ransomware*, pois a alternativa seria pagar o resgate, o que não é indicado, já que acaba incentivando a prática de desenvolvimento do *malware,* além de não existir a certeza que o pagamento seja uma garantia de retorno dos dados criptografados.

2.3 Técnicas de análise de *malware*

Segundo Malin *et. al.* (2012), cinco fases são utilizadas para identificar *malwares*: (1) preservação forense e exame dos dados voláteis; (2) exame da memória do dispositivo infectado; (3) análise forense propriamente dita, com a verificação dos dispositivos de armazenamento permanentes, como por exemplo, o disco rígido; (4) verificação das características do arquivo desconhecido e pré análise estática; e (5) aplicação das **técnicas de análises estática e dinâmica do arquivo suspeito.**

A quinta fase executa as análises estáticas e dinâmicas aproveitando as informações coletadas, caso possível, da fase anterior. Nesta fase são executadas as ferramentas de diagnóstico e elaborados relatórios técnicos das análises.

Sikorski (2012) conceitua a **análise estática** como o processo de análise do código ou da estrutura de um programa para determinar a sua função. A análise é feita sem que o arquivo seja executado no sistema. Subdivide-se em dois tipos.

A primeira, a **análise estática básica** utiliza técnicas de identificação como o uso de programas de antivírus, análises de *hashes*, verificação de cabeçalho de arquivos suspeitos e funções. O uso de programas de antivírus é o recurso mais utilizado na análise estática básica por ser facilmente encontrado para aquisição e por ser amplamente divulgado.

Já a segunda, a **análise estática avançada** executa a técnica de engenharia reversa, que analisa o arquivo suspeito e extrai comandos do código fonte para estudo, o que é extremamente (SGANDURRA *et. al.*, 2016).

Segundo Sikorski (2012), na análise dinâmica observa-se o arquivo e baseado em seu comportamento determina-se sua função. A análise dinâmica possui dois tipos.

No primeiro, **análise dinâmica básica**, o arquivo suspeito é executada em um ambiente virtualizado e suas rotinas e instruções de execução são monitoradas, permitindo a construção automática de um conjunto de características das instruções que estão sendo executadas. Em alguns casos, é utilizada a observação do pré e do pós execução do código para definir se o arquivo suspeito é benigno ou maligno. Sikorski (2012), diz que toda a análise baseia-se na observação das API.

Sikorski (2012) define um conjunto de APIs que podem ser observadas conforme Quadro 1 no Apêndice A.

A técnica de análise dinâmica requer um ambiente seguro, controlado e isolado, onde as instruções do arquivo suspeito possam ser executadas e registradas sem causar qualquer tipo de dano. Para isso, utiliza-se a técnica de ambiente confinado, conhecida como *sandbox*.

Já no segundo, **análise dinâmica avançada,** utiliza-se a técnica de ***debugging*** **(depuração)** para verificar o código do arquivo suspeito, geralmente **modo *assembly*.** Esse modo é mais utilizado quando se quer realizar a análise de *malwares* pois não requer acesso ao código fonte original do arquivo suspeito.

2.3.1 Técnicas de evasão

Branco *et. al.* (2012), divide essas técnicas em quatro categorias:

1. **Ofuscação**, consiste em dificultar ou até mesmo impedir a análise estática através da busca por assinaturas em um banco de dados;
2. ***Antidisassembly***, essa técnica tem como objetivo impedir a realização da engenharia reversa impedindo ou atrasando sua detecção.
3. **Anti-VM,** essa técnica detecta o uso de máquinas virtuais e *sandbox,* assim *malwares* não se executam nesses ambientes impedindo sua identidicação e detecção.
4. **Antidebugging**, o *malware* identifica quando há algum programa que realiza o *debugg* do código. Ao identificar o *debbuger,* o *malware* passa a controlar o fluxo das execuções de suas instruções.

3 TRABALHOS RELACIONADOS

Nesta seção são apresentados os trabalhos que tratam sobre o assunto de identificação e classificação de *ransomware* que fundamentam a presente dissertação com a utilização do uso de análise estática e/ou análise dinâmica.

3.1 Análise Estática

Esta subseção apresenta os trabalhos que exploram o uso da técnica de análise estática. Estes trabalhos apresentam expressiva contribuição a esta pesquisa, principalmente, quanto a maturidade e evolução que apresentam na implementação da técnica de análise estática.

3.1.1 Detecção de algoritmos criptográficos

A dissertação de Hosfelt (2015) apresenta a proposta de realizar a detecção automática e classificação de algoritmos criptográficos em programas binários por meio do aprendizado de máquina supervisionado.

Hosfelt (2015) propõe criar programas com a função de executar determinados algoritmos criptográficos com a implementação do OpenSSL para cada algoritmo. Os arquivos criados são gerados a partir dos algoritmos criptográficos Advanced Encryption Standard (AES) no modo Cipher Block Chaining (CBC), Rivest Cipher 4 (RC4), **R**ivest, **S**hamir e **A**dleman (RSA), Secure Hash Algorithm (SHA1), Message-Digest algorithm 5 (MD5), Triple Data Encryption Standard (3DES) nos modos CBC, Cipher FeedBack (CFB) e Output FeedBack (OFB). Os modelos são treinados com base nos algoritmos criptográficos criados. Todos os programas geram criptografia e decriptografia. Para gerar dados suficientes para o treinamento Hosfelt (2015) compilou os programas no GNU Compliner Collection (GCC) e no Clang (https://clang.llvm.org) junto com outros programas que não possuem o recurso de criptografia. Com isso são geradas 317 amostras de programas binários.

Após criar os programas utiliza-se a ferramenta DBI Tool para poder avaliar o comportamento do programa em tempo de execução e analisar, durante o tempo de injeção de instruções, o comportamento por meio da técnica de análise estática.

O DBI é uma ferramenta que realiza o *debugging* (depuração) do arquivo analisado em modo ***assembly.*** Isso torna possível monitorar e controlar a quantidade de vezes que uma determinada instrução é executada assim que o programa é carregado na

memória RAM. Esta ferramenta é capaz de quantificar e classificar em categorias as instruções em linguagem de máquina, como *NOP* (não execute nenhuma operação), *SYSCALL* (efetua chamada ao sistema), *JP* (salta para um endereço), *SLEEP* (entra em modo de espera), ADD (adiciona valor a algum registro ou contador), SUB (subtrai valor a algum registro ou contador) e *MOV* (Mover) por exemplo. Se a instrução indicar um *loop* e esse *loop* indica a presença de criptografia é possível que o arquivo seja um *malware.* Em suas observações Hosfelt (2015) afere oito instruções que executam *loops* constantes.

Hosfelt (2015) implementa três tipos de modelos para análise: (1) o **modelo de detecção** que tem o objetivo de classificar as amostras como tendo ou não criptografia; (2) o **modelo de classificação de tipo** que classifica o que está presente nos programas: *hashing* ou criptografia; e (3) o **modelo de classificação de algoritmo** que tenta identificar o tipo de criptografia aplicada.

Os experimentos de Hosfelt (2015) são executados em uma máquina virtual com 1 núcleo e 1024 MB de memória. Os algoritmos supervisionados SVM, Naive Bayers e árvore de decisão foram avaliados de duas formas: (1) validação cruzada dentro do conjunto de dados do original; e (2) na validação de um conjunto de dados separado. Já para avaliar os algoritmos não supervisionados são aplicados três métricas: (1) homogeneidade; (2) integridade; e (3) *v-score.* E é utilizado o *framework* scikit-learn, com a linguagem de programação Python (versão não informada), para treinar os três modelos definidos e os 317arquivos compilados em GCC (https://gcc.gnu.org) e Clang, como já informado.

O algoritmo que apresenta melhor performance nos experimentos de Hosfelt (2015) é o SVM. Segundo a autora, o algoritmo de classificação e detecção apresenta a precisão de 95%. No entanto, de acordo com a autora, a pequena similaridade e quantidade de amostras pode ter influenciado o resultado. Entretanto, o trabalho pode ser utilizado em conjunto com o método de análise dinâmica para aumentar a precisão da detecção.

3.1.2 Ferramenta para detecção e classificação de *ransomware* em dispositivos com sistema operacional Android

Andronio (2012) desenvolve uma ferramenta chamada Heldroid, para dispositivos móveis com sistema operacional Android, versão não informada, que é capaz de detectar e classificar o *ransomware* (proposta principal, impedir o contágio e disseminação) e remover o *ransomware* (proposta secundária).

O autor informa que aplicar a técnica de análise dinâmica em dispositivos móveis resultaria em um diagnóstico lento. Por este motivo, a técnica de análise estática com o aprendizado de máquina supervisionado foi utilizada para o desenvolvimento da ferramenta. Para o treinamento foram utilizadas cento e oitenta e cinco mil amostras (arquivos contaminados e não contaminados) o que forma o total de 232 conjuntos de exemplos.

Após realizar engenharia reversa verificando um número consistente de famílias de *ransomware* conhecidos e suas variantes, Andronio (2012) conclui que três características distinguem o *ransomware* de outros *malwares*: (1) a ameaça ao usuário; (2) o bloqueio do dispositivo; e (3) a criptografia dos arquivos. As duas últimas características, bloqueio e a criptografia do arquivo estão incorporados ao código fonte, o que permite procurar por sinais suspeitos utilizando a análise do programa através da técnica de análise estática.

Para realizar a tarefa proposta a ferramenta Heldroid precisa executar um conjunto de ações:

1. **Filtragem:** Heldroid realiza um conjunto de ações com o uso da técnica de análise estática.
   - 1.1 **Consulta:** é realizada uma consulta à base de dados pública de amostras de *malware* Contágio Mini-Dump. Caso o arquivo esteja cadastrado ele é rotulado como contaminado;
   - 1.2 **Descompactação e análise dos binários:** os arquivos são descompactados do instalador (apk) com a ferramenta APKTool – versão 2.00 rc3 (https://ibotpeaches.github.io/Apktool/). E com as ferramentas Luyten (https://github.com/deathmarine/Luyten) e dex2jar (https://sourceforge.net/projects/dex2jar/) é realizada a engenharia reversa com o objetivo de obter os binários para análise;
   - 1.3 **Análise heurística:** os comandos são verificados por meio da análise heurística.
2. **Detecção de ameça de texto.** Considerada como o núcleo do sistema, a ferramenta mapeia cadeias de texto simples para uma representação matemática para a classificação. Como o texto pode estar escrito em múltiplos idiomas é utilizada a biblioteca de código aberto Cybozu Langdetect (https://github.com/shuyo/language-detection/tree/master/src/com/cybozu/labs

/langdetect) para detectar o idioma automaticamente. Caso seja detectado algum texto ameaçador o arquivo é marcado como contaminado.

3. **Deteção de criptografia.** Este módulo busca comandos que possam indicar sequências de *loops* nos binários extraídos do processo de filtragem 1.2 – descompactação e análise dos binários. Caso encontre fluxos de execução como leitura, criptografia, exclusão e armazenamento o arquivo é rotulado como contaminado.
4. **Detecção de Bloqueio**. Este módulo observa a saída (resultado) do processo de filtragem 1.3 – análise heurística. A ferramenta realiza uma verificação nos comandos aferidos e no arquivo de instalação, AndroidManifest.xml. Caso encontre algum comando de bloqueio de tela o arquivo é marcado como contaminado.

Apesar da ferramenta de Andronio (2012) utilizar a técnica de análise estática para detecção e classificação de *ransomware,* Heldroid pode executar a análise dinâmica com o uso da ferramenta de *sandbox* Andrubis (http://anubis.iseclab.org). No entanto, o autor informa que tal técnica em dispositivos móveis causa uma lentidão de quatro a seis minutos para o diagnóstico.

Segundo Andronio (2012), Heldroid é a primeira aplicação de detecção de *ransomware* para dispositivos móveis que utilizam sistema operacional Android e que através de aprendizado de máquina com um conjunto pequeno de amostras *ransomware,* foi capaz de identificar novas famílias e suas variantes. Ainda, foi capaz de detectar duzentas e sete dos duzentos e trinta e dois exemplos de *ransomware* analisados, sendo que cento e noventa e quatro foram detectadas por análise estática e somente treze por análise dinâmica.

O trabalho de Andronio (2012) foi categorizado como análise estática pois, grande parte da execução da ferramente, executa recursos nesse modo. O modo de análise dinâmica foi utilizado quando o Heldroid foi incapaz de detectar e classificar os arquivos suspeitos. Vale ressaltar que a análise dinâmica é executada manualmente. Caso o arquivo suspeito não fosse detectado na análise estática, provavelmente o dispositivo teria sido infectado em um ambiente de produção, já que muitos usuários de dispositivos móveis não possuem o conhecimento técnico para executar a análise dinâmica.

3.1.3 Comparação de estudos para identificar e classificar *malware* ATP

Nath e Mehtre (2014) realizam uma análise preditiva com foco nos métodos que um invasor usaria para comprometer um sistema a Advanced Persistent Threat (APT) -

ataques direcionados e persistentes. Entre os exemplos deste tipo de ataque podem ser citados: (1) Stuxnet, descoberto em 2010, este *malware* tinha como objetivo atacar o Supervisory Control and Data Acquisition (SCADA), utilizado para controlar as centrífugas de enriquecimento de urânio iranianas; e (2) Operação Aurora, que atacou grandes empresas de tecnologia como a Google, Yahoo, Adobe, Symantec e Juniper Networks.

Segundo os autores, a análise dinâmica pode ser facilmente enganada já que alguns *malwares* podem detectar a presença de um emulador ou ambiente virtualizado, portanto, seria difícil prever o comportamento real do arquivo suspeito.

Diversos métodos de mineração de dados ou aprendizado de máquina são usados para a análise estática de *malware* e o principal obstáculo está na ausência de uma quantidade de amostras de treinamento suficiente já que esse tipo de análise é realizada em *malwares* do tipo não APT, segundo os autores. Assim, Nath e Mehtre (2014) realizam um estudo comparando cinco implementações de pesquisadores e observam qual método é o mais indicado para a classificação de *malware.*

1. **Kolter e Maloof (2004)** desenvolvem o método de analisar a sequência de *bytes* de arquivo, chamada de *n*-Gram. O *n* representa o número de *bytes* para formar uma única amostra e para determinar o melhor algoritmo de aprendizado de máquina. Os autores realizam um *benchmarking* entre Naive Bayes, Árvore de Decisão (C4.5) e AdaBoost (versão aprimorada da Árvore de Decisão). O melhor desempenho é na utilização do AdaBoost, que obtém uma TFP 0,98 para uma TFP de 0,05 com 1651 *malwares* e 1971 arquivos legítimos;

2. **Lyda at el.** desenvolvem uma ferramenta que utiliza a técnica de sequência de *byte de* arquivo, chamada de *bintropy*. Essa técnica pressupõe que se um arquivo é criptografado, então sua sequência de *bytes* seria muito alterada, o que aumenta o conteúdo das informações do arquivo. Este método é utilizado para identificar o *malware* que utilização criptografia, como é o caso do *ransomware*. Os experimentos mostram que o valor da entropia seria alto se o arquivo fosse criptografado. A partir da análise de 21.576 arquivos os autores desenvolvem uma métrica de entropia que poderia ser generalizada para qualquer tipo de arquivo executável com o tamanho máximo de 500 KB;

3. **Santos et. al. (2013)** utilizam um método de aprendizado de máquina semi-supervisionado com o algoritmo ROC-SVM. Para realizar a classificação de *malware* o autor verifica em 67 *malwares* quantas vezes um *Operation Code* (OPCODE) se repete e quais são os OPCODEs coincidentes. OPCODE é uma única instrução que pode ser executada pela Unidade Central de Processamento – C*entral Process Unit* (CPU). Em linguagem de máquina é um valor binário ou hexadecimal, que é carregado no registro de instrução. No mnemônico de linguagem de montagem, um OPCODE é um comando como MOV, NOP, ADD ou JP usado em linguagem assembly, por exemplo. Neste caso, é verificado quantas vezes o arquivo executa um comando de *looping* ou de exclusão de pastas ou arquivos;

4. **Bilar (2007)** utiliza dados estatísticos para detecção de executáveis maliciosos. O autor recorre à frequência de OPCODE, ou seja, verifica quantas vezes um determinado comando é executado em 67 amostras *malwares* e considera os de maiores frequência como maliciosos;

5. **Shafiq at. el. (2009)** apresenta o sistema PE-Probe que utiliza a técnica Portable Executable (PE) Header*.* O PE Header é a técnica que extrai informações do cabeçalho do arquivo executável para formar um conjunto de classificadores de aprendizado de máquina. Essa técnica supõe que o comportamento do programa pode ser predito analisando seu cabeçalho, onde são armazenadas as informações das bibliotecas Dynamic Link Library (DLL) e os arquivos de vínculo. O sistema PE-Probe funciona em dois níveis: (1) **detector de pacote**, classifica em compactado e não compactado. A classificação do cabeçalho é realizada através da análise das informações extraídas das seções: padrão, não padrão, executável, leitura ou gravação, entradas na tabela de endereços de importação Import Address Table (IAT), código e dados. A classificação é realizada usando a classe multi-layer perceptron (MLP) que utiliza o aprendizado de máquina supervisionada; (2) **avaliação das chamadas do cabeçalho do arquivo**, as APIs e componentes de vínculo são avaliados para verificar se existe alguma chamada ou vínculo suspeito.
   Para verificar qual algoritmo apresenta melhor precisão para classificação é realizado um *benchmarking* entre cinco algoritmos de aprendizagem: (1) aprendizado baseado em instância; (2) árvore de decisão; (3) Naive Bayes; (4)

Repeated Incremental Pruning to Produce Error Reduction (RIPPER); e (5) *Support Vector Machine* (SVM). A árvore de decisão apresenta o melhor resultado diante dos demais algoritmos. Shafiq at. el. (2009) afirma que sua ferramenta apresenta ter melhor precisão e desempenho do que a abordagem do *n*-Gram, que é mais adequada para a classificação de dados vagamente estruturados, e não adequada para explorar informações estruturais específicas do formato de um arquivo PE.

A Tabela 2 apresenta uma comparação das informações apresentadas de cada um dos autores estudados por Nath e Mehtre (2014).

Tabela 2: Tabela de comparação de pesquisadores com o uso de análise estática do estudo de Nath e Mehtre (2014).

Pesquisador	Malware	Legitimo	Algoritmo	Recurso	Verdadeiro Positivo	Falso Positivo
Kolter e Maloof	1651	1971	AdaBoost	N-gram	0.98	0.05
Lyda et. al.	21567	0	Bintropy	Sequencia de byte	-	0.005
Santos et. al.	17000	-	ROC-SVM	OPCODE	0.96	0.05
Bilar	67	-	Frequência	OPCODE	-	-
Shafiq at. el.	-	-	Árvore de Decisão	Cabeçalho do Arquivo Executável	0.996	0.003

Fonte: Nath e Mehtre (2014).

Nath e Mehtre (2014) concluem: (1) nenhuma das pesquisas concentram uma solução para detectar *malware* do tipo APT e recomendam o desenvolvimento de alguma ferramenta, mesmo que teórica, de aprendizado de máquina que correlacione as atividades de um arquivo benigno de um arquivo maligno; (2) apesar dos estudos realizados, nenhum trata especificamente de *ransomware* que criptografam arquivos.

A utilização da técnica de análise estática para detecção de APT, pode ser utilizado como um recurso quando se conhece o arquivo a ser analisado. No entanto, o método pode ser testado com a integração ao método de análise dinâmica para verificar se os resultados aprimoram o método de classificação.

Conforme os autores descrevem, alguns *malwares* possuem a habilidade de diagnosticar recursos como ambientes virtualizados e emuladores. Para evitar que o *ransomware* realize esse diagnostico o trabalho proposto nesta dissertação implementa técnicas que evitam essa análise.

O estudo dos autores é de grande contribuição para este trabalho, já que foi possível utilizar conceitos, cálculos e especificidades de alguns *malwares.* Além disso, o estudo é específico na busca de informações estruturais, ou seja, com análise estática é possível verificar quais atributos são mais qualificados para serem utilizados por

algoritmos de aprendizagem de máquina. Mesmo que o estudo não seja específico sobre *ransomware*, é possível utilizar esses atributos como escolha para constituir vetores que serão utilizados pelos algoritmos de aprendizagem nesta dissertação.

3.2 Análise Dinâmica

Esta subseção apresenta os trabalhos que exploram o uso da técnica de análise dinâmica. Com exceção de Lutz (2008), todos os trabalhos aqui relacionados apresentam a implementação da técnica de *sandbox*. Já o trabalho de Lutz (2008), apresenta uma correlação, através do desenvolvimento de uma ferramenta, entre códigos criptográficos e sua alocação na memória RAM.

3.2.1 Uso de análise dinâmica para detectar e classificar *ransomware*

Sgandurra et. al. (2016) propõem o desenvolvimento de uma ferramenta que trabalha de forma a auxiliar o programa de antivírus, EldeRan. EldeRan possui a finalidade de detectar e classificar um *ransomware*. O desenvolvimento da ferramenta baseia-se na utilização da técnica de análise dinâmica e o recurso de *sandbox stand-alone*, Cuckoo Sandbox, com Windows XP SP2.

Os autores afirmam que a análise estática utilizando o mecanismo de detecção por assinatura apresenta um baixo índice detecção de *malware* e estão propensos a serem enganados justificando que em um conjunto de 3000 amostras detectadas através de um vetor de infecção do tipo *phishing,* somente 6% foram cadastradas no *site* VirusTotal, especializado na detecção de *malwares,* devido a sofisticadas estratégias de evasão, de encapsulamento e a grande quantidade de variantes espalhadas na rede mundial de computadores.

Sgandurra et. al. (2016) descrevem as principais técnicas de disseminação do *ransomware*, que são: *e – mail spam, phishing, exploit kits, downloader, trojan botnets*, táticas de engenharia social e sistemas de distribuição de tráfego *Traffic Distribution System* (TDS).

Os métodos utilizados para recuperação dos dados em caso de infecção por *ransomware* são:

- **Reinicializar o equipamento em modo de segurança** e restaurar o sistema e/ou executar o programa de antivírus. Esse recurso só pode ser utilizado para o *ransomwares* do tipo *locker;*

- **Pagar o resgate.** Não se recomenda realizar qualquer tipo de pagamento pois isso acaba incentivando a prática de desenvolvimento do *malware,* além de não se ter a certeza que o pagamento seja uma garantia de retorno dos dados criptografados;
- **Restaurar os arquivos de um b*ackup.*** Neste caso é necessário que exista uma rotina de cópia para eventuais desastres.

Caso os arquivos já tenham sido criptografados é computacionalmente quase impossível conseguir a chave privada. Existem casos em que os desenvolvedores do *ransomware* acabam deixando a chave privada escrita no código fonte. Nesses raros casos, a engenharia reversa seria útil para decifrar o código.

Para os autores, *ransomwares* possuem características dinâmicas únicas e para evitar sua propagação é crucial diagnosticar suas variantes durante a sua primeira aparição. Segundo os autores, uma possível solução é a implementação de um algoritmo baseado em aprendizado de máquina que detecte e classifique o *malware*.

Sgandurra *et. al.* (2016) desenvolvem usando como ferramenta algoritmo de aprendizado de máquina baseado em regressão logística e utiliza para treinamento um conjunto de 582 amostras de *ransomware* do *site* VirusShare (https://virusshare.com) de 11 classes diferentes e em sua maior parte do tipo *crypto* e 942 amostras de arquivos considerados benignos (este conjunto de amostras retirados do sistema operacional como arquivos com extensão .zip, .jpeg, .pdf, programas de desenvolvimento, jogos, ferramentas de escritório, entre outros). Essas amostras foram submetidas a teste de verificação por antivírus e *sites* especializados em segurança e rotulados como *ransomwares* e benignos.

Os autores submetem os arquivos para análise dinâmica com o Cuckoo Sandbox e geram um relatório com as características de cada tipo de arquivo: *ransomware* ou benigno. Este relatório foi então convertido para o formato JavaScript Object Notation (JSON) para criar uma matriz para ser utilizado pela EldeRan.

Para testar eficiência da ferramenta um conjunto de arquivos benignos e malignos (exclusivamente *ransomwares*) foram avaliados para detecção e classificação. Os autores realizam um *benchmark* entre EldeRan e os classificadores SVM linear e Naive Bayes. Nesta comparação o EldeRan superou o classificador SVM linear e Naive Bayes, que ficou em última posição noteste.

Sgandurra et. al. (2016) comparam EldeRan com cinco tipos de fornecedores de antivírus e em todos os casos a ferramenta os superou:

- 96,3% de detecção de *ransomware*;

- 93,3% de detecção de novas famílias de *ransomware*;
- Taxa de erro de 2,4% na classificação. Esta é uma taxa considerada baixa quando comparada com a taxa do site VirusTotal, especializado em segurança.

O trabalho de Sgandurra *et. al.* (2016) supera as expectativas com relação a detecção de *ransomware*, utilizando parte de sua metodologia de desenvolvimento com outros algoritmos de aprendizado de máquina e a análise estática, espera-se criar uma matriz que supere a detecção e classificação conseguida pelos autores.

3.2.2 Análise e extração de características estruturais e comportamentais de *Malware*

Caldas (2016) realiza um estudo dos meios existentes para classificação de *malwares* em geral. Como forma de contribuição ele desenvolve um meio de extrair as características estruturais e comportamentais e determina um perfil para cada tipo de *malware.*

Segundo o autor, para determinar o perfil do *malware* é necessário descobrir o seu objetivo. Para cada perfil, elencam-se quais são os comportamentos mais comuns apresentados por amostras de *malware.* Esses comportamentos são utilizados para nortear a escolha de características, estruturais e comportamentais, das amostras a serem extraída e analisada.

Caldas (2016) determina nove perfis de *malware* baseados em seus objetivos:

- *Banker:* roubar dados bancários;
- *DoS Bot:* ataque DoS;
- *Ransomware:* sequestrar arquivos;
- *Damageware:* causar danos a diposítivos específicos;
- *Backdoor*: abrir portas;
- *Downloader:* realizar download de *malware;*
- *Adware:* exibir propaganda;
- *Spyware:* roubar informações.

Ransomware, para o autor, é o *malware* com objetivo de restringir (bloqueando e/ou criptografando) o acesso ao sistema infectado, e exigir o pagamento de uma quantia de dinheiro para que o acesso seja restabelecido, como forma de resgate. O *ransomware* possui duas características comportamentais: (1) acessa uma grande quantidade de arquivos em um curto período de tempo; e (2) cria arquivos criptografados de tamanho excessivo.

Para extrair as características do *malware,* Caldas (2016) utiliza as características estruturais e comportamentais do arquivo através da técnica de análise dinâmica, *sandbox standalone*. As características estruturais são aquelas que podem ser extraídas diretamente do arquivo como o tamanho do binário, número de seções com nome desconhecido e chamadas de APIs. Características comportamentais são aquelas que podem ser observadas durante a execução do arquivo como por exemplo, autoexecução, exclusão do arquivo original, criar variantes polifórmicas, número de arquivos acessados, chamadas de APIs, porcentagem de pacotes de saída e se conectar a *hosts* suspeitos.

Para realizar a extração das características comportamentais e estruturais Caldas (2016) utiliza o Cuckoo SandBox. Após a coleta das informações o autor estrutura o resultado e associa cada característica a um tipo de *malware.*

Observando-se os dados, percebe-se que a única característica que pode ser promissora para classificação de ransomware é a de acesso a diversos arquivos em um curto período de tempo, segundo o autor. Vale ressaltar que o fato dessa ser a única característica associada à *ransomware* deve-se ao fato das amostras selecionadas serem do tipo *crypto*. Caso alguma aplicação de criptografia legítima fosse submetida a essa avaliação, possivelmente seria interpretada como um *ransomware.*

O próprio autor considera pequena a quantidade de amostras sendo necessário um conjunto maior para ampliar o estudo. O estudo pode ser ampliado com a aplicação de algum algoritmo de aprendizado de máquina para automatizar a classificação junto a análise de arquivos legítimos.

3.2.3 Um estudo de caso com o uso de análise dinâmica para compreender *malwares*

Schmidt et. al. (2014) criam um cenário com o intuito de implementar a quinta fase da metodologia de análise de *malware*, a técnica de análise dinâmica com o recurso de *sandbox standalone* em um ambiente corporativo. O cenário tem como objetivo apresentar o diagnóstico de um determinado arquivo, se é benigno ou maligno.

Os autores utilizam para a máquina *host* o sistema operacional Ubuntu 64 *bits* com o programa de virtualização VirtualBox que virtualiza o sistema operacional Microsoft Windows XP.

A operação da análise segue três etapas: (1) arquivo é selecionado e enviado para o ambiente virtual; (2) o arquivo passa por análises; e (3) diagnóstico.

Os autores selecionam 2 arquivos para submissão para a máquina *host,* que executa o Cuckoo SandBox. O primeiro arquivo é selecionado em uma caixa de e-mail

com o nome de **Nota Fiscal 00356280011.zip**. Schmidt et. al. (2014) ressaltam que o meio de propagação foi a técnica de *phishing*, já que o e-mail veio com o nome de assunto relacionado ao envio de nota fiscal de algum serviço contratado, o que despertou o interesse do usuário (vítima). E que o arquivo passou pelo *antispam*, o que significa que ele possui uma assinatura confiável. Já o segundo arquivo é capturado na rede local com o nome de **1DB6476C766555C9996B25D19F97B9BC.EXE**

O trabalho de Schmidt et. al. limita-se a submeter os arquivos suspeitos para a máquina virtualizada para ser analisada pelos recursos do Cuckoo SandBox. As análises executadas pelo Cuckoo SandBox foram:

1. Verificação do arquivo no *site* VirusTotal. O Cukoo SandBox possui uma função integrada de submissão do arquivo para o *site*, onde é possível verificar se o arquivo possui uma assinatura cadastra como *malware* em algum banco de dados de antivírus;

2. São realizados os testes para gerar relatórios:

2.1 Detalhes dos arquivos analisados: nesta seção são incluídos as assinaturas dos arquivos, tipo, nome, extensão, tamanho, os *hashes* MD5, SHA1, SHA256, SHA512, o hash SSDEEP, quando possível também a assinatura PeiD;

2.2 *Screenshot* da execução da máquina: durante o processo de análise são gerados várias cópias da tela;

2.3 Análise do m*alware disassembler*: o analisador captura as bibliotecas que o arquivo submetido utiliza, endereço de memória, informações sobre seus dados. Essas informações são importantes para descobrir qual o seu real objetivo;

2.4 Sistema de arquivos: realiza o monitoramento das alterações realizadas durante a análise;

2.5 Atividades de rede: neste teste é capturado o tráfego de rede;

2.6 Análise de comportamento: este teste monitora a ação em tempo de execução do arquivo submetido;

2.7 Análise dos processos e resultados: nessa etapa são apresentados relatórios detalhando se o arquivo submetido é um *malware* ou benigno.

Os resultados mostram uma série de arquivos alterados e criados, registros do sistema operacional Microsoft Windows modificados com mais de trezentos acessos e alterações e alguns *screenshot* capturados. Com base nessas informações foi possível constatar que os arquivos submetidos eram *malware*.

O trabalho de Schmidt et. al. apresentou um cenário, utilizando a análise dinâmica detalhando a operação do programa Cuckoo SandBox. Nesta dissertação, é utilizada a mesma técnica e a mesma ferramenta para gerar o vetor para ser utilizado na unificação das duas análises (estática e dinâmica) e aplicar o aprendizado de máquina supervisionado.

3.2.4 Uso da técnica de mancha para identificar criptografia em reserva de memória *random access memory* (RAM)

Lutz (2008) apresenta uma ferramenta, desenvolvida em Valgrid, até o momento exclusivo para sistemas operacionais Gnu/Linux, que analisa automaticamente as entradas de código de criptografia no momento da alocação de memória e a técnica de análise de mancha com o uso da técnica de análise dinâmica em redes de computadores.

Para isso Lutz (2008) extrai um conjunto de informações do binário em tempo de execução dos algoritmos de criptografia Blowfish, Twofish, OpenSSL e Gnu Privacy Guard (PG), tais como: informações padrões de acesso à memória e ao fluxo de controle, incluindo os circuitos de programas e chamada de funções.

Em uma segunda etapa, buscam-se nos dados extraídos características convencionais do processo de criptografia, como por exemplo: o elevado número de operações aritmética com números inteiros. Ainda são monitorados *buffers* com a técnica de análise de mancha para buscar reservas de memória com características do processo de criptografia. Propõem-se a utilização de heurística para observar códigos com *loops*, diretamente com a análise comportamental.

O autor utiliza o pacote Wine (https://www.winehq.org) para testar se a ferramenta identifica a aplicação, se é legítima ou se trata de um programa malicioso. Para teste é implementado o *malware botnet* Kraken. O Wine é um programa capaz de executar aplicações do sistema operacional Windows em sistemas operacionais GnuLinux, os programas são executados como se fossem nativos do sistema operacional hospedeiro sem a necessidade de uso programas de virtualização, com por exemplo o Virtualbox, reduzindo o consumo de computacionais, como, por exemplo, memória RAM.

O programa desenvolvido por Lutz não quebra a criptografia do *malware,* ela se aproveita das informações extraídas no processo de análise dinâmica para verificar o endereçamento de memória utilizada. Caso o endereçamento seja similar aos utilizados pelo processo de criptografia dos algoritmos de criptografia Blowfish, Twofish, OpenSSL e

GnuPG e não tenha sido executado pelo usuário do dispositivo, é possível dizer se o programa trata-se, ou não, de um *malware*.

Segundo o autor, a ferramenta demonstra eficiência na detecção de programas que utilizam algoritmos criptográficos com reserva de memória no tráfego de rede.

Uma das limitações da ferramenta, afirma Lutz (2008), é o fato de ser executada somente em sistemas operacionais Gnu/Linux, o que limita as avaliações e experimentos.

3.3 Comparação

Esta subseção dedica-se a comparar as principais referências relacionados ao tema deste trabalho com o intuito de apresentar as semelhanças, diferenças e contribuições para o presente. A Tabela 3 apresenta dados comparativos que tem por objetivo posicionar este trabalho a partir do relacionamento de um conjunto de critérios relevantes à execução da proposta desta dissertação.

Tabela 3: Comparativo de trabalhos.

Referência	1	2	3	4	5	6	7	8	9
Hosfelt (2015)	-	A	A	-	A	-	A	A	-
Andronio (2012)	-	A	A	-	-	P	A	-	A
Nath e Mehtre (2014)	A	A	A	-	A	-	A	-	-
Sgandurra (2016)	-	A	-	A	A	A	-	-	A
Caldas (2016)	-	P	P	A	-	A	-	-	P
Schimidt et. al. (2014)	-	-	-	A	-	A	-	-	-
Lutz (2008)	-	A	-	A	-	-	-	-	-
Proposta deste trabalho	-	A	A	A	A	A	A	A	A

Legenda:

Recursos	**1**	*Malware(s)* do tipo ATP.
	2	Identificação de criptografia.
	3	Executa a análise estática.
	4	Executa a análise dinâmica.
	5	Realiza a classificação utilizando algoritmo(s) de aprendizado de máquinas.
	6	Uso de SandBox.
	7	Uso de ferramenta para técnica de engenharia reversa.
	8	Utiliza *framework* de aprendizagem de máquinas.
	9	Detecção e classificação de *ransomware.*
Aplicação	**A**	Aplicado
	P	Aplicado Parcialmente.
	-	Não aplicado.

Fonte: Elaborado pelo próprio autor.

O trabalho de Nath e Mehtre (2014) é o único que se propõe a realizar um estudo que busque soluções para *malwares* do tipo ATP. Os autores concluem que objetivo proposto não é alcançado, no entanto, a pesquisa apresenta dados relevantes que contribuem para este trabalho, principalmente em relação à extração do cabeçalho do arquivo e OPCODE.

Dos sete trabalhos pesquisados seis realizam detecção de criptografia, sendo que Caldas (2016) não foca especificamente na busca desse recurso, já que seu trabalho é mais generalizado procurando identificar todo tipo de *malwares*, correlacionando suas atividades com seus objetivos específicos com o uso da análise dinâmica. Hosfelt (2015), Sgandurra (2016), Lutz (2008) e Nath e Methre (2014) focam em detectar criptografia com o uso das técnicas de análise estática ou dinâmica para detectar criptografia em computadores. Já Andronio (2012) foca em buscar criptografia em dispositivos móveis, especificamente Android. A busca por criptografia é um recurso essencial para este trabalho, já que *ransomware* pode criptografar arquivos.

Os trabalhos de Sgandurra (2016) e Andronio (2012) são os únicos que se dedicam a detectar exclusivamente o *malware* do tipo *ransomware*, sendo que Andronio (2012) detecta em dispositivos móveis enquanto Sgandurra (2016) foca detectar em computadores.

Hosfelt (2015) e Sgandurra (2016) são os únicos a utilizar a detecção e classificação com o uso de algoritmo de aprendizagem de máquinas, o que demonstra que esse tipo de recurso ainda é pouco utilizado. Sendo Sgandurra (2016) o único a utilizar algoritmo de aprendizagem de máquinas com a finalidade de detectar *ransomware.* A Tabela 2 apresenta Nath e Mehtre (2014) como utilizadores deste tipo de metodologia, no entanto, ressalta-se que esse recurso foi utilizado por autores pesquisados em seu trabalho.

SandBox foi utilizado nos trabalhos de Sgandurra (2016), Caldas (2016) e Schimidt et. al. (2014) como método de detecção de *malwares.* A utilização deste recurso por Andronio (2012) foi caracterizada como parcial pois o foco de seu trabalho é realizar a detecção de *ransomware* por análise estática (a análise dinâmica foi utilizada de forma manual quando houve falha na detecção por análise estática).

Com a exceção de detecção de *malwares* do tipo ATP, o presente trabalho é o único que apresenta uma proposta que correlaciona os demais recursos apresentados na Tabela 3. A principal contribuição é que este trabalho propõe a combinação da utilização das técnicas de análise estática e dinâmica, identificação de criptografia, uso de

classificadores de algoritmos de aprendizagem de máquinas, ferramentas de sandbox e engenharia reversa e *framework* de aprendizagem de máquinas para a identificação e classificação de *ransomware.*

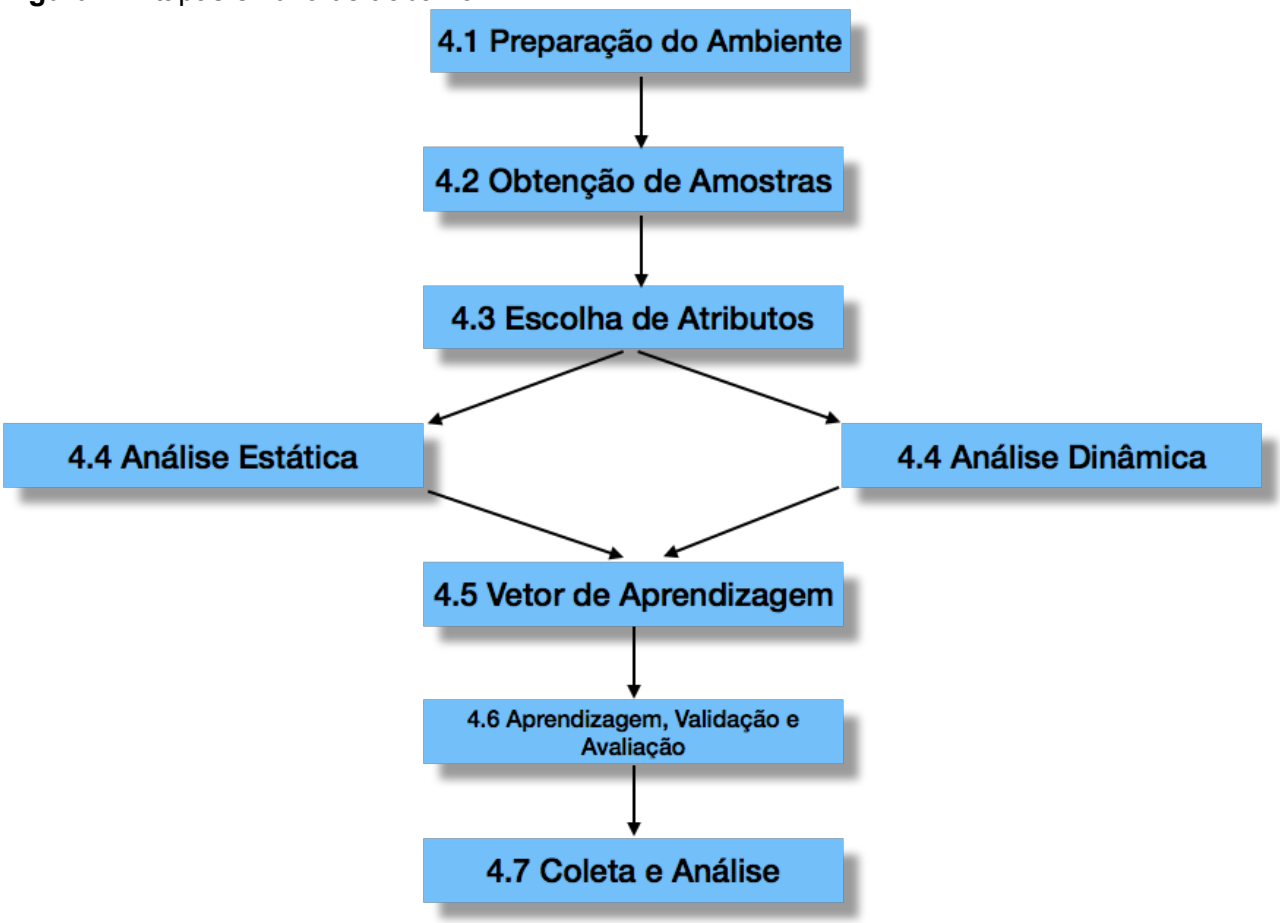
A partir dos trabalhos analisados nesta seção, é apresentada na Seção 4 uma proposta para unificação das técnicas de análise estática e dinâmica para automatizar o processo de classificação de *ransomware* com o uso de algoritmo de aprendizado de máquina.

4 PROPOSTA DE UNIFICAÇÃO E APRENDIZADO DE MÁQUINA

Esta seção descreve a proposta de unificação das análises estática e dinâmica que através do aprendizado de máquina supervisionado deve classificar se um determinado arquivo é um *ransomware* ou é benigno.

A proposta deste trabalho é divida em sete etapas, conforme a Figura 7.

Figura 7: Etapas e fluxo de trabalho.



Fonte: Elaborado pelo autor.

A Figura 7 apresenta as etapas e o fluxo das atividades que são executadas neste trabalho.

4.1 Preparação do Ambiente

A Figura 8 apresenta o ambiente de trabalho.

Figura 8: Ambiente de trabalho.

Fonte: Elaborado pelo próprio autor.

Como pode ser observado na Figura 8, a máquina hospedeira é um MacBook Pro meados de 2012, com processador Core i7, 8 GB de memória RAM, Hd de 500 GB Solid State Drive (SSD), placa gráfica HD Intel e sistema operacional macOS High Sierra (versão 10.13.1). Os programas instalados relevantes para este trabalho são o Cuckoo Server 2.0.5, VirtualBox 5.2.0, Python 3.6.3, scikit-learn 0.19.1, Volatily 2.6, o pacote Tcpdump, a biblioteca M2Crypto e o programa de desenvolvimento Anaconda (https://anaconda.org) 5.0.1 gráfico. O Volatily 2.6, o pacote Tcpdump e o M2Crypto são pré-requisitos para a operação do Cuckoo Server 2.04. O Volatily é uma ferramenta que realiza análise forense em *dumps* de memória. O pacote Tcpdump realiza captura de rede durante a execução do arquivo analisado. Já a biblioteca M2Crypto é utilizada dar suporte a diversos tipos de algoritmos criptográficos.

O Cuckoo Sandbox é uma ferramenta de código aberto desenvolvido em Python, assim, para utiliza-lo é necessário que Python esteja instalado tanto no dispositivo hospedeiro quanto nos dispositivos que são alvos das análises estática e dinâmica. Considerando o requisito de utilização do Python para a análise de *malware*, a opção aqui é utilizar esta linguagem para desenvolver as predições em aprendizado de máquina. O scikit-learn é o principal *framework* de aprendizado de máquina utilizado em Python.

A máquina hospedeira possui as seguintes funções:

- Manter as máquinas virtuais em um ambiente isolado, controlado e seguro para que possam ser alvo das análises estáticas e dinâmicas;
- usar o Cuckoo Sandbox para enviar aquivos para as máquinas virtuais para as análises estática e dinâmicas;
- Gerar os relatórios das análises estática e dinâmica;
- Converter os relatórios em um vetor de atributos para o algoritmo de aprendizado de máquina;

- Executar o aprendizado de máquina por meio do Python (scikit-learn). São utilizados os algoritmos SVM, o Naive Bayes e a árvores de decisão;
- Validar e avaliar qual algoritmo teve melhor desempenho. Para validar os experimentos é utilizado o método de validação cruzada e para aferir o desempenho de cada algoritmo são aplicados os métodos de concordância Kappa, acurácia, revocação e precisão.

A escolha de SVM, Naive Bayes e árvore de decisão como algoritmos de aprendizado de máquina que são comparados, deve-se ao fato destes algoritmos destacarem-se em pesquisas anteriores.

O SVM é utilizado no trabalho de Santos *et. al.* (2013) e no de Nath e Metre (2014) para classificação de *malware*. Shafiq *at. el.* (2009) também classifica malwares. Para isso utiliza árvores de decisão para análise estática e obtém melhor precisão na comparação com outros algoritmos de aprendizagem de máquina. O algoritmo SVM obtém melhor desempenho quando comparado a Naive Bayes e árvores de decisão no trabalho de Hosfelt (2015).

As máquinas virtuais VM1 e VM2, representadas na Figura 8 simulam um computador com Microsoft Windows XP Pro *Service Pack* – SP 3, com processador de 2 núcleos, memória de 2 GB, HD virtual dinamicamente alocado de 40 GB.

A função das máquinas VM1 e VM2 é simular um ambiente de produção de um usuário corporativo ou doméstico. Para que a simulação fique mais próxima da realidade são instalados os programas de escritório Office 2007, Adobe Acrobat Reader 11.o, 7zip 17.01, Microsoft .Net 3.5 e Python 2.7.14 (pré-requisito do Cuckoo Sandbox). A fim de evitar interferência, não são instalados programas de antivírus, *firewalls* e as atualizações automáticas do Microsoft Windows XP são desabilitadas.

Para que haja interação ente as máquinas virtuais e a máquina hospedeiro as seguintes configurações são necessárias:

- O arquivo do Cuckoo agent.py localizado na máquina hospedeira em ~/bin/cuckoo/Agent deve ser copiado para as máquinas VM1 e VM2, dentro da pasta do python em c:\Python27;
- O adaptador de rede no VirtualBox precisar ser habilitado e configurado como placa de rede exclusiva de hospedeiro. Esta configuração garante o confinamento e isolamento das máquinas VM1 e VM2.

Para análise estática é utilizado o programa pescanner.py. O pescanner.py é uma ferramenta de código aberto desenvolvida em Python que extrai características estruturais de arquivo utilizando a engenharia reversa, como por exemplo, data de criação e API.

O Cuckoo Sandbox é o programa responsável por realizar a análise dinâmica. Seus pré-requisitos de instalação são os pacotes libmagic, cairo, pango e openssl. O pacote Homebrew é instalado porque o sistema hospedeiro é um MacBook Pro com sistema operacional macOS. Homebrew é um gerenciador de pacotes não oficial para sistemas operacionais macOS, bastante similar aos gerenciadores de pacotes apt-get e aptitude utilizados em Gnu/Linux.

Outras bibliotecas opcionais e pacotes não obrigatórios são instalados para deixar o sistema ajustado às necessidades desde trabalho:

- **Jinja2:** biblioteca para a renderização de relatórios HTML em interface *web;*
- **Dpkt:** biblioteca para a extração de arquivos com extensão .pcap (dados de tráfego de rede);
- **Magic*:*** para identificação de formato de arquivos;
- **Pymongo:** para armazenamento de resultados no banco de dados MongoDB;
- **Pefile:** para análise estatística de binários do tipo Portable Executable (PE32) no Microsoft Windows;
- **Libvirt:** para o uso do gerenciador Kernel Virtual Machine (KVM);
- **Bottlepy:** para utilizar a ferramenta api.py (chamadas de API) e web.py (servidor *web* para apresentação de resultados);
- **Voletility:** para análise forense de memória;
- **MAEC Python *bindings*:** para relatórios *Malware Attribute Enumeration and Characterization* – MAEC;
- **Chardet:** para detecção de codificação de *strings;*
- **Pydeep e ssdeep:** para calcular o *fuzzy hash* dos arquivos;
- **Yara e yara python:** para verificar assinaturas equivalentes em assinaturas;
- **Tcpdump:** para capturar e analisar tráfego de rede.

O componente dll.py, localizado em ~data/analyzer/windows/modules/packages/ é responsável por execuções e análises e possui um parâmetro chamado *loader*. Esse parâmetro é usado para trocar o nome do processo de DLL, por padrão rundll32, para impedir técnicas de *anti-sandboxing*. Assim toda vez que um *malware* verificar a DLL

carregada ele não consegue evadir-se da análise. Este componente se inicializa automaticamente com o Cuckoo Sandbox conforme a Figura 9.

Figura 9: Inicialização de componentes do Cuckoo Sandbox

```
[cuckoo.core.startup] DEBUG:      |-- AntiAnalysisDetectFile
[cuckoo.core.startup] DEBUG:      |-- AntiAVDetectFile
[cuckoo.core.startup] DEBUG:      |-- AntiAVDetectReg
[cuckoo.core.startup] DEBUG:      |-- AntiAVSRP
[cuckoo.core.startup] DEBUG:      |-- AntiDBGDevices
[cuckoo.core.startup] DEBUG:      |-- AntiDBGWindows
[cuckoo.core.startup] DEBUG:      |-- AntiSandboxFile
[cuckoo.core.startup] DEBUG:      |-- AntiSandboxForegroundWindow
[cuckoo.core.startup] DEBUG:      |-- AntiSandboxIdleTime
[cuckoo.core.startup] DEBUG:      |-- AntiSandboxRestart
[cuckoo.core.startup] DEBUG:      |-- AntiSandboxSleep
[cuckoo.core.startup] DEBUG:      |-- AntiVMBios
[cuckoo.core.startup] DEBUG:      |-- AntiVMComputernameQuery
[cuckoo.core.startup] DEBUG:      |-- AntiVMCPU
[cuckoo.core.startup] DEBUG:      |-- AntiVMDiskSize
[cuckoo.core.startup] DEBUG:      |-- AntiVMIDE
[cuckoo.core.startup] DEBUG:      |-- AntiVMSCSI
[cuckoo.core.startup] DEBUG:      |-- AntiVMServices
[cuckoo.core.startup] DEBUG:      |-- AntiVMSharedDevice
```

Fonte: CuckooSandbox

Como visto na Figura 9, o componente AntiSandboxFile (dll.py) é inicializado automaticamente, assim como outros componentes que evitam a técnicas de evasão de análise.

4.2 Obtenção das Amostras

As amostras de *ransomware* são obtidas do site VirusShare e de um conjunto de amostras coletadas em ambientes de produção corporativa do próprio autor, assim são obtidas amostras no formato Windows PE32. Os arquivos possuem as extensões .htm, .html, .jpeg, .jpg, .xls, .pdf, .ico, .docx, .exe, .msi, .zip, .arj, .rar, .wma, .wav, .ogg, .mp3, .mp4, .aiff, .ra, .mov, .flv, .mkv, .mkt, .rm, .avi, .3gp, .cam e .divx.

Para garantir que os arquivos selecionados sejam *malwares* do tipo *ransomware* os arquivos são enviados para análise em dois *sites* especializados em segurança o Metadefender e VirusTotal.

Para compor as amostras de arquivos benignos são selecionados arquivos de:

- Ambiente de produção corporativa de servidores Microsoft Windows Server Standard 2012 R2, coletados das pastas c:\windows, c:\windows\system e c:\windows\system32;

- Arquivos dos sites sourceforge (sourceforge.net) e baixaki (www.baixaki.com.br). Ambos escolhidos por terem um vasto repositório de arquivos para *download* gratuitamente;
    - Arquivos do computador do autor.

As amostras de arquivos benignos tem o mesmo formato e extensões das amostras de *ransomware.* A opção de selecionar somente arquivos do tipo PE32 deve-se ao fato de criar uma padronização para análise. Assim são coletadas o total de 589 amostras de arquivos benignos.

Assim como é feito para a população de arquivos *ransomwares* os arquivos selecionados como benignos são submetidos para análise nos mesmos *sites* de segurança, assim garante-se que essa população não contenha nenhum tipo de *malware,* mesmo os compactados (arj e zip, por exemplo) ou aqueles que possuem encapsulamento (exe e msi, por exemplo).

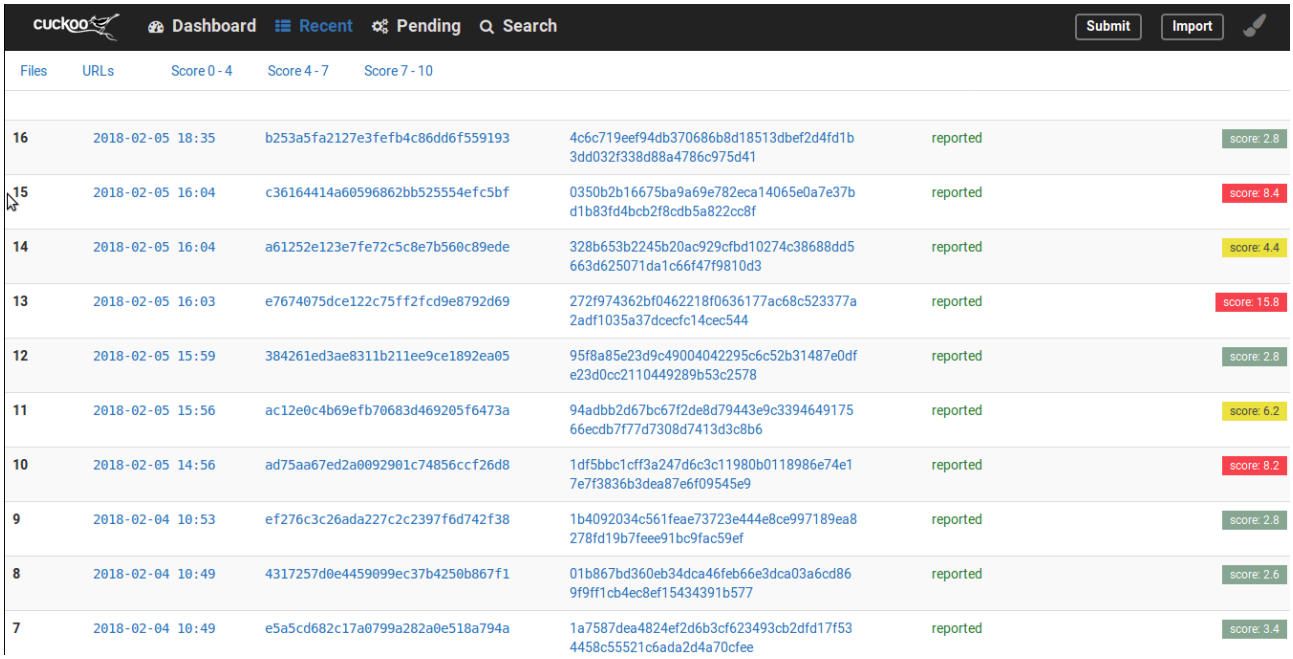
4.3 Escolha de Atributos

Nesta etapa são selecionados os atributos para a análise dos arquivos *ransomware* e dos arquivos considerados benignos.

Este trabalho utiliza o método de ranqueamento de atributos para selecionar os atributos relevantes para a classificação pelos algoritmos de aprendizagem de máquina e somente a população de *ransomware* é submetida a essa técnica de seleção. O método de ranqueamento baseia-se em classificar os atributos em posições. A medida que um determinado atributo aparece em uma amostra analisada este atributo ganha um ponto. Ao final da análise da população selecionada, os atributos são posicionados e assim é possível determinar quais atributos são relevantes para a classificação.

O Cuckoo Sandbox utiliza o sistema de pontuação para determinar se um determinado arquivo submetido é *malware* ou benigno. Em caso de *malware* o arquivo é pontuado diferente de zero para determinar seu grau de periculosidade, conforme pode ser visto na Figura 10.

Figura 10: Relatório de uma determinada quantidade de *ransomware* submetidos ao Cuckoo Sandbox



Fonte: Cuckoo Sandbox.

A Figura 10, apresenta uma relação de *ransomwares* submetidos para a análise e os classifica em três categorias:

- Pontuação entre zero e quatro: são considerados arquivos com baixa periculosidade*;*
- Pontuação entre 4 e 7: são considerados arquivos com médio grau de periculosidade;
- Pontuações acima de 7: são considerados arquivos com alto grau de periculosidade. Os arquivos pontuados acima de 10.0 pontos são considerados críticos, ou seja, possuem alto grau de dano computacional.

Este trabalho utiliza a pontuação do Cuckoo Sandbox para realizar uma nova categorização dos *ransomwares.* A opção de re-categorizar os *ransomwares* faz-se necessária para prevenir os fenômenos de *overfitting* e *underfitting.* Assim as amostras são separadas em quatro categorias:

- Categoria 1: pontuações entre 0.1 a 3.5, com 67 ransomwares;
- Categoria 2: pontuações entre 3.6 a 6.0, com 83 *ransomwares*;
- Categoria 3: pontuações entre 6.1 a 8.8, com 83 *ransomwares*;
- Categoria 4: pontuações acima de 8.9, com 48 *ransomwares.*

O processo de ranqueamento de atributos inicia-se com a extração dos dados estruturais e comportamentais através das análises estática e dinâmica nas quatro categorias.

Em ambas as análises os relatórios são tratados para gerar dados consistentes para a seleção de atributos. Para essa função é utilizada a biblioteca Pandas para estruturar, organizar as informações em um *dataframe* e remover as informações desnecessárias.

A análise estática é realizada com a utilização do pescanner.py em toda a população de *ransomwares* mantendo os resultados dos relatórios divididos nas quatro categorias mencionadas anteriormente. De cada categoria são selecionados os cinco atributos que se destacam através do sistema de ranqueamento. A consolidação desses atributos gera sete atributos, conforme a Tabela 4.

Tabela 4: Tabela de atributos para a análise estática.

EPEiD	DateSusp	EPSusp	SuspEntropy
IATSusp	SuspiciousString	CRCSusp	

Fonte: Elaborado pelo próprio autor.

Como pode ser visto na Tabela 4, sete atributos são previamente selecionados para a análise estática. Cada atributo da análise estática possui função específica, conforme:

- EPEiD: tem por finalidade verificar se algum cifrador, empacotador ou compilador conhecido por desenvolver *malware* é utilizado. Caso seja encontrado é atribuído o valor 1 para o atributo, caso contrário é atribuído o valor 0;
- DateSusp: verifica se a data de criação é suspeita ou não. A data é considerada suspeita se o ano for menor que 2000 e maior que a data atual. Caso seja suspeita é atribuído o valor 1 e quando for considerada normal é atribuído 0;
- EPSusp: verifica se o ponto de entrada da seção PE é suspeita. Um ponto de entrada de uma seção é o nome de uma seção PE que contém o AddressOfEntryPoint. Em geral, para aquivos legítimos ou não empacotados, o AddressOfEntryPoint fica em uma seção nomeada .code ou .text para arquivos em modo usuário. Já para arquivos de *drivers* de *kernel* se estabelece na seção PAGE ou INIT. Se o AddressOfEntryPoint é considerado suspeito é atribuído o valor 1 para o atributo, caso contrário o valor 0;
- SuspEntropy: este atributo deriva do trabalho de Lyda (2007), que identifica seções que possuem uma entropia muito alta ou muito baixa. A entropia é um valor entre 0 e 8. Um arquivo com entropia maior que 7 é considerado suspeito. Quando a entropia é

considerada suspeita é atribuído o valor 1 para o atributo e 0 quando for considerada normal;

- IATSusp: esse atributo realiza buscas na tabela de endereços de arquivos importados. Existe referência para as seguintes APIs: OpensProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, ReadProcessMemory, CreateProcess, WinExec, ShellExecute, HttpSendRequest, InternetReadFile, InternetConnect, CreateService e StartService. Caso haja referência o atributo recebe o valor 1, caso contrário recebe o valor 0;
- SuspiciousString: verifica se existe no código desmontado alguma *string* considerada suspeita. Quando é encontrada alguma *string*, a ela é atribuido o valor 1 e quando não for considerado suspeita atribui-se o valor 0;
- CRCSusp: a verificação cíclica de redundância – *Cyclic Redundancy Check* (CRC) é o responsável em verificar se um arquivo está intacto, se possui empacotamento ou se ele foi alterado. Ou seja, se foi adicionado algum código ou arquivo. Quando o CRC é considerado suspeito é atribuído o valor 1, caso contrário atribui-se o valor 0.

Já para análise dinâmica é utilizado o Cukoo Sandbox. Por padrão, o Cuckoo Sandbox armazena os dados das análises em banco de dados Mongodb. O Mongodb permite exportar dados nos formatos .JSON e .csv.

Para compor os atributos para o método de ranqueamento são extraídas as informações referentes às chamadas de APIs do relatório. As chamadas de APIs do Microsoft Windows são armazenadas no banco de dados chamado cuckoo, coleção *analysis* e atributo apistats da referida ferramenta, conforme a Figura 11.

Figura 11: Relação de chamadas APIs de um *ransomware* após análise dinâmica realizada pelo Cuckoo Sandbox


```
"apistats" : { "1472" : { "NtOpenSection" : 15, "GetForegroundWindow" : 28,
"getaddrinfo" : 5, "SetFileTime" : 18, "GetFileVersionInfoSizeW" : 2,
"GetFileAttributesW" : 131, "GetVolumePathNamesForVolumeNameW" : 4, "RegEnumKeyExA" : 500,
"RegOpenKeyExW" : 1496, "SearchPathW" : 44, "InternetCrackUrlA" : 10, "SetErrorMode" :
145, "GetFileInformationByHandle" : 1, "RegOpenKeyExA" : 986, "HttpSendRequestW" : 1,
"GetCursorPos" : 321, "GetUserNameW" : 1, "FindWindowExA" : 1, "GetUserNameA" : 5,
"FindResourceExW" : 229, "NtCreateFile" : 87, "GetSystemTimeAsFileTime" : 84,
"GlobalMemoryStatusEx" : 1, "InternetSetOptionA" : 14, "SetWindowsHookExA" : 8,
"LoadResource" : 272, "CoInitializeSecurity" : 1, "SetFileAttributesW" : 13,
"CoGetClassObject" : 20, "NtQueryInformationFile" : 21, "RegCreateKeyExW" : 41,
"DeviceIoControl" : 14, "InternetSetStatusCallback" : 1, "NtQueryKey" : 3,
"OpenServiceA" : 2, "RegQueryValueExA" : 380, "LookupPrivilegeValueW" : 2,
"NtQueryValueKey" : 147, "getsockname" : 3, "RegQueryValueExW" : 1243, "CreateActCtxW" :
6, "NtDeviceIoControlFile" : 10, "NtReadFile" : 559, "RegEnumValueA" : 33, "NtWriteFile" :
15, "LdrGetDllHandle" : 407, "NtQuerySystemInformation" : 13, "CreateThread" : 12,
```


Fonte: Banco de dados cuckoo, coleção *analysis* e atributo apistats

A Figura 11, apresenta as chamadas APIs realizadas por um *ransomware.* Como feito na análise estática, os relatórios são armazenados separadamente em categorias. De cada categoria são selecionados 20 atributos principais, que consolidados geram o total de trinta e quatro, conforme a Tabela 4.

Tabela 5: Tabela de atributos para a análise dinâmica.

CreateFile	CreateMutex	CreateProcess	CreateRemoteThread
CreateService	OpenFile	DeleFile	FindWindow
OpenMutex	OpenSCManager	ReadFile	ReadProcessMemory
RegDeleteKey	RegEnumKeyEx	RegEnumValue	CreateSection
RegOpenKey	ShellExecute	TerminateProcess	URLDownloadToFile
WriteFile	WriteProcessMemory	ZwMapViewOfSection	LocalDll
GetProcedureAddress	OpenKey	QueryValueKey	IsDebuggerPresent
GetSystemMetrics	CreateMutant	OpenSection	RegQueryValueEx
RegCloseKey	OpenMutant		

Fonte: Elaborado pelo próprio autor.

A Tabela 4 apresenta a relação dos atributos selecionados através do método de ranqueamento executado na análise dinâmica. Cada um dos atributos da análise dinâmica (chamadas API) possui uma função específica:

- CreateFile: cria um arquivo ou abre um arquivo existente;
- CreateMutex: cria uma exclusão mútua de objeto;
- CreateProcess: cria e inicia um processo no Microsoft Windows;
- CreateRemoteThread: cria e/ou inicia uma *thread* em um processo remoto. Normalmente utilizado para injetar código em um processo existente;
- CreateService: cria um serviço. *Malwares* utilizam esta API para iniciar um serviço no *boot* do sistema operacional e carregar outros aplicativos e/ou APIs;
- OpenFile: abre um arquivo;
- DeleFile: exclui um arquivo;
- FindWindow: realiza uma busca por janelas abertas;
- OpenMutex: abre um *handle* para um objeto com exclusão mútua. *Handle* é um número que o Microsoft Windows atribui para cada objeto que está sendo manipulado;
- OpenSCManager: abre um *handle* para o gerenciador de controle de serviços. *Malwares* comumente utilizam essa API para interagir com os processos dos serviços do Microsoft Windows, assim é possível iniciá-los e pará-los;
- ReadFile: lê um arquivo;

- ReadProcessMemory: está API possui a finalidade de ler a memória de um processo remoto. Endereços de memória podem ser copiados utilizando esta API;
- RegDeleKey: apaga chaves e/ou subchaves de registro;
- RegEnumKeyEx: enumera as subchaves de um registro;
- RegEnumValue: enumera os valores de uma chave de regitro;
- CreateSection: cria um objeto de seção;
- RegOpenKey: abre um *handle* com objetivo de controlar a leitura e edição de um registro;
- ShellExecute: utilizado para executar programas;
- TerminateProcess: finaliza processos e suas *threads*;
- URLDownloadToFile: realiza *download* de arquivos;
- WriteFile: grava dados no dispositivo de saída;
- WriteProcessMemory: grava dados em um processo remoto. *Malwares* utilizam essa API para injetar código;
- ZwMapViewOfSection: mapeia a visão de uma seção em um espaço de endereçamento virtual;
- LocalDll: função de baixo nível que carrega uma DLL em um processo. Em casos de estudos de *malwares* indica que um programa atua de forma evasiva;
- GetProcedureAddress: recupera o endereçamento de uma função carregada na memória. *Malwares* utilizam essa API para importar funções de outras DLLs e adicionar às importadas no cabeçalho do arquivo PE;
- OpenKey: abre um *handle* para controlar a leitura e edição de um registro;
- QueryValueKey: acessa valores de uma chave de registro;
- IsDebuggerPresent: verifica se um determinado processo está sendo depurado. *Malwares* utilizam esta API para verificar se algum depurador está em operação e assim evadir-se;
- GetSystemMetrics: obtém informações de configuração do sistema. Utilizado para injetar código para sistemas específicos;
- CreateMutant: um objeto mutante é criado e é aberto um *handle* para ele;
- OpenSection: abre um *handle* para uma seção;
- RegQueryValueEx: retorna o tipo e o valor para o nome específico associado a uma chave de registro;
- RegCloseKey: fecha o *handle* de uma chave de registro;

- OpenMutant: abre um *handle* para o objeto para execução exclusiva da instância do *malware.*

4.4 Análises

Nesta etapa a população de *ransomware* e a população de arquivos benigno são submetidas às análises estática (*script* pescanner.py e dinâmica com o objetivo de gerar valor para os atributos selecionados na etapa anterior.

O Cuckoo SandBox S*erver*, instalado na máquina hospedeira possui um *script* de submissão de arquivo para a máquina virtual VM2. A máquina VM2 possui o arquivo agent.py que recepciona o arquivo submetido pela máquina hospedeira e realiza a coleta de informações para compor o relatório da análise dinâmica. Ao final da análise, um arquivo com os resultados é gerado e valores são atrelados aos atributos.

Ressalta-se que as análises da população de *ransomware* são submetidas na mesma ordem de análise, tanto para a análise estática quanto para a análise dinâmica, assim o índice de identificação atribuída para uma determinada amostra submetida à análise estática será igual à identificação atribuída quando essa mesma amostra é atribuída à análise dinâmica.

4.5 Vetor de Aprendizagem de Máquina

Os relatórios gerados na etapa anterior precisam ser tratados para gerar um arquivo a ser submetido aos algoritmos de aprendizagem de máquina. Nesta etapa os relatórios são carregados em um *dataframe* em Python para serem tratado pela biblioteca Pandas (comando: *import* pandas.).

Os relatórios das análises estática e dinâmica da população de *ransomwares* são lidos pelo Pandas como um *dataframe* e definidos com o nome da variável df_estatica_ransomware, enquanto o relatório de análise dinâmica, também lido como um *dataframe*, é definido com o nome da variável df_dinamica_ransomware. Os *dataframes* podem ser lidos com o comando: df_estatica_ransomware.head().

Os atributos relacionados nas Tabelas 4 e 5 são os atributos que possuem relevância para a classificação e que são utilizados pelos algoritmos de aprendizagem de máquina. Assim, os demais atributos, que são lidos como colunas pelo Pandas, são removidos dos *dataframes* com o comando: **df_estatica_ransomware.drop**

(‘coluna_a_remover’, axis = 1, inplace = True) e **df_dinamica_ransomware.drop (‘coluna_a_remover’, axis = 1, inplace = True)**, com exceção do atributo ID.

Como os *data*frames possuem uma coluna de identificação – ID que representa a amostra analisada pelas análises estática e dinâmica. Realiza-se a unificação dos *dataframes* com a finalidade de criar um *dataframe* consolidado, com o comando: **pandas.merge(df_estatica_ransomware, df_dinamica_ransomware, how=’inner’, on=’key’)**, esse novo *data frame* é atribuído à variável df_ransomwares.

O método *merge* do Pandas realiza mesclagem das informações baseadas em uma chave comum em ambos os *dataframes*, neste caso a coluna ID.

Com os atributos consolidados adiciona-se uma coluna ao *dataframe* df_ransomwares com o nome de classe e a todos os registros desse novo atributo adiciona-se o valor 1. O Pandas não trabalha com palavras (chamadas de *strings* em linguagem de programação), assim admite-se que todo o registro com valor 1, dentro do atributo classe é classificado como *ransomware.* Para adicionar a coluna classe ao *dataframe* df_ransomware utiliza-se o comando: df_ransomware[‘Classe’] = 1.

Já para a população de arquivos benignos, realiza-se os seguintes procedimentos:

1. Importam-se os relatórios das análises estáticas e dinâmicas para serem lidos pelo Pandas como *dataframe*;

2. Atribui-se para a análise estática a variável df_estatica_benignos e para a análise dinâmica a variável df_dinamica_benignos;

3. São removidos os atributos não relacionados nos Quadros 4 e 5, com exceção do atributo ID;

4. Unificam-se os *dataframes* df_estatica_benignos e df_dinamica_benignos e é criado um novo *dataframe:* e a esse novo *dataframe* atribui-se uma variável chamada df_benignos;

5. Adiciona-se uma nova coluna ao *dataframe* df_benignos com o nome de Classe e atribui-se o valor de 0 a todos os registros. Assim admite-se que todo registro, do atributo classe, com valor 0 é classificado como *benigno.*

Como ambos *dataframes* df_benignos e df_ransomware possuem os mesmos atributos representados pelas colunas é possível realizar a concatenação com o comando: pandas.concat([df_benignos, df_ransomware], axis=0) e a atribuição da variável df_analises.

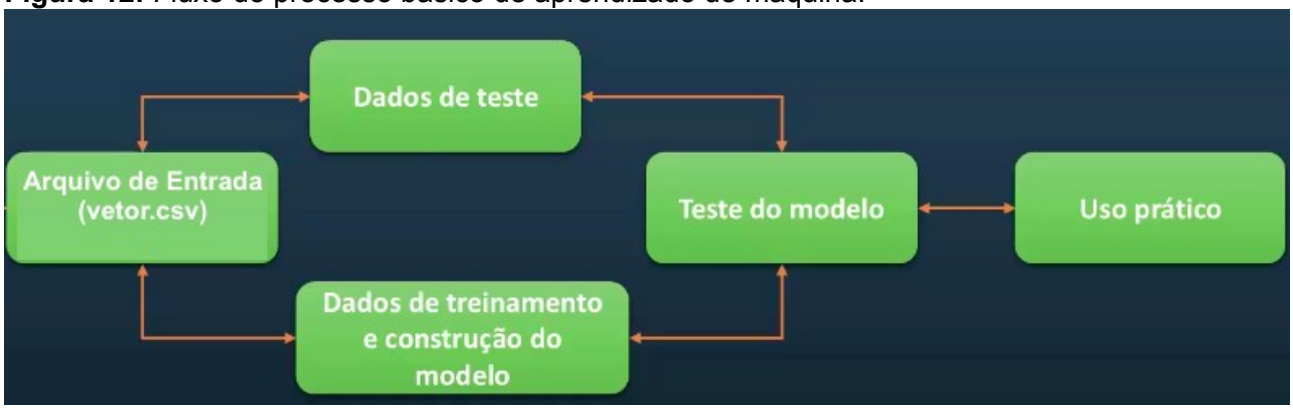
O df_análises possui o atributo ID que não é relevante para a classificação, assim deve-se removê-lo. O *dataframe* resultante é utilizado como vetor de classificação pelos algoritmos de aprendizagem de máquina. Para exportar esse *dataframe* em arquivo utiliza-se o comando: pandas.to_csv (‘vetor1.csv’, sep=’;’, decimal=’,’).

O *dataframe* exportado em arquivo com o uso da biblioteca Pandas possui os atributos relevantes aos algoritmos de aprendizagem de máquina dos *ransomwares* e dos arquivos benignos. Este arquivo está tratado e pronto para ser submetido aos algoritmos de aprendizagem de máquina e aos processos de validação e avaliação do modelo que é gerado.

4.6 Aprendizagem de Máquina, Validação e Avaliação

O arquivo gerado na etapa anterior (vetor.csv) é utilizado como vetor de aprendizado de máquina assim, diversos algoritmos de classificação podem ser testados. A Figura 10 apresenta o fluxo do processo básico de aprendizado de máquina que é utilizado nesta etapa do trabalho.

Figura 12: Fluxo do processo básico de aprendizado de máquina.



Fonte: Elaborado pelo autor.

Como pode ser visto no fluxo de processo representado na Figura 10 o arquivo **vetor.csv**, é utilizado como entrada neste processo. Ressalta-se que este arquivo contém dados adquiridos na etapa 4.2 e que são considerados prontos para uso pois as informações irrelevantes para os algoritmos de aprendizagem de máquina foram removidas na etapa 4.5. Este arquivo é divido em dois processos denominados dados de treinamento e construção do modelo e dados de teste:

Dados de treinamento e construção do modelo é o processo que utiliza a parte da divisão do arquivo de entrada para construir o modelo do algoritmo de aprendizagem escolhido. Ou seja, neste processo o algoritmo encontra os parâmetros que faz com que o modelo consiga predizer o que ele aprendeu com os dados de treinamento.

Dados de teste é a segunda parte arquivo de entrada, utilizado no processo de teste do modelo. Estes dados não são utilizados para treinar o modelo.

É importante que o conjunto de dados de testes sejam descorrelacionados do conjunto de dados de treinamento pois é nesse instante que se verifica se o modelo é efetivo na realização de predições (classificações) de novos dados.

No processo de **Teste do modelo** os dados são testados apresentando informações de como o modelo aprende e se este aprendizado é correto. Neste ponto verifica-se o modelo construído consegue capturar as informações relevantes para uma determinada amostra a partir do conjunto de dados de testes.

Os processos de entrada de arquivo, treinamento e construção, dados de testes e testes do modelo são repetidos até que o modelo que obtenha a precisão e a consistência desejada para que seja utilizado de forma prática.

Os algoritmos de aprendizagem de máquina escolhidos neste trabalho são o SVM, o Naive Bayes e a árvore de decisão. O método de validação cruzada é utilizado para obter os resultados dos experimentos.

A avaliação experimental de um algoritmo de aprendizado de máquina pode ser realizada segundo diferentes aspectos, tais como precisão do modelo gerado, a compreensibilidade do conhecimento extraído, o tempo de aprendizado e alguns requisitos de armazenamento do modelo, entre outros (CARVALHO *et. al.*, 2011).

Para verificar o desempenho da metodologia utilizada são aplicados os métodos de concordância Kappa, acurácia, revocação e precisão**.**

O processo de **uso prático** deste trabalho engloba a utilização do *framework* scikit-learn que é utilizado para realizar as predições, avaliação e validação dos modelos criados. Nos três experimentos os algoritmos são executados na modalidade de validação cruzada a fim de obter resultados estatisticamente significativos.

São realizados três experimentos com o intuito de submeter o vetor gerado na etapa 4.5 aos algoritmos de aprendizagem de máquina selecionados. Para garantir a isonomia deste trabalho adota-se o critério de dividir o vetor de aprendizado em treino e teste, sendo que 70% dos dados do vetor são utilizados para treino e os demais 30% para teste. A seleção e divisão dos dados de treino e teste é feita de forma aleatória.

Para isso é necessário importar um método do scikit-learn que realiza essas divisões e definir variáveis para os objetos de treino e de teste. Abaixo segue os comandos relacionados a essas funções comentados, os comentários são antecedidos pelo símbolo de cerquilha (#).

```
#importar o arquivo com o pandas e atribuir a variável pd_vetor.
    pd_vetor('vetor.csv', sep=';', decimal=',')

#importar método scikit-learn que divide o vetor em treino e teste.
    from sklearn.model_selection import train_test_split

#especificar os atributos para a classificação para a variável x.
    x = pd_vetor([Colunas_de_atributos])

# definir o que é classificado para a variável y.
    y = pd_vetor(['Classe'])
#definir 70% dos dados para treino e 30% para teste de forma aleatória.
    train_test_split(x, y, test_size=0.3, random_state=101)
```

No **primeiro experimento realizam-se** classificações com o algoritmo (SVM), no **segundo experimento** o algoritmo usado é a árvore de decisão e por fim, no **terceiro** e último **experimento** avalia-se o vetor de aprendizagem de máquina com o algoritmo Naive Bayes.

5 COLETA E ANÁLISE DE RESULTADOS

Nesta seção são apresentados os resultados da aplicação dos algoritmos de aprendizagem de máquina ao arquivo vetor.csv. Também é realizada uma comparação entre os algoritmos baseados nos métodos de avaliação precisão revocação, acurácia e kappa.

O primeiro experimento com o uso do algoritmo de aprendizado de máquina SVM gera os seguintes dados como resultado da submissão do arquivo vetor.csv:

- 126 amostras classificadas como VP;
- 63 amostras classificadas como VN;
- 40 amostras classificadas como FP;
- 38 amostras classificadas como FN.

Já o segundo experimento que utiliza o algoritmo de árvore de decisão gera os seguintes resultados:

- 134 amostras classificadas como VP;
- 68 amostras classificadas como VN;
- 36 amostras classificadas como FP;
- 29 amostras classificadas como FN.

Por último, o terceiro experimento gera os seguintes dados com o uso do algoritmo de aprendizagem de máquina Naive Bayes:

- 129 amostras classificadas como VP;
- 91 amostras classificadas como VN;
- 29 amostras classificadas como FP;
- 18 amostras classificadas como FN.

Utilizando o método de avaliação Kappa e os métodos precisão, acurácia e revocação baseados na matriz de confusão a Tabela 5 apresenta o resultado consolidado dos experimentos.

Tabela 6: Resultado consolidado dos experimentos.

Fonte: Elaborado pelo autor.

A Tabela 5 apresenta os resultados dos experimentos realizados na etapa 4.6 e pode-se constatar que o algoritmo Naive Bayes apresenta o melhor desempenho dentre os analisados. Os resultados aferidos demonstram que o algoritmo se sobressai sobre os demais nas quatros avaliações propostas. Sendo que o método de avaliação Kappa utilizando o índice de concordância Kappa proposto por Landis e Kock (1977) conforme Tabela 1, apresenta um resultado considerado como concordância substancial.

6 CONSIDERAÇÕES FINAIS

O assunto r*ansomware* representa um desafio significativo para os pesquisadores e especialistas em segurança de TI. Com rápida disseminação, desenvolvimento de novas variantes e sofisticadas técnicas de ofuscamento e evasão o estudo dos *ransomwares* requer que pesquisadores apresentem constantes abordagens que acompanhem suas técnicas evolutivas.

Este trabalho auxilia nesse esforço, apresentando uma abordagem diferenciada comparada aos trabalhos relacionados. Os métodos tradicionais abordam o uso isolado ou, em alguns casos, em pares dos métodos de análise estática, análise dinâmica e aprendizagem de máquina.

O foco deste trabalho é unificar os métodos de identificação de *ransomware* (análise estática e análise dinâmica) na criação de um vetor de treinamento gerado com o uso do *framework scikit*-learn. Este vetor é utilizado para comparar desempenho de alguns algortimos de aprendizado de máquina (SVM, Naive Bayes e árvore de decisão).

Ao fim dos experimentos, coleta dos dados e análise observa-se que o algoritmo de aprendizagem de máquina Naive Bayes obteve o melhor desempenho com taxas acima de 80%, quando são utilizados os métodos de avaliação baseados na matriz de confusão acurácia, precisão e revocação, e concordância substância quando analisado ao método de avaliação Kappa utilizando o índice de concordância Kappa proposto por Landis e Kock (1977).

Para trabalhos futuros, sugere-se comparar as técnicas de análise estática e dinâmica isoladamente para classificação de *ransomware*, ou seja, utilizar os relatórios das análises, tratar os dados e gerar um vetor de aprendizagem e comparar cada uma das análises com a análise unificada e assim submeter aos algoritmos de aprendizagem de máquina utilizados neste trabalho.

Ainda, recomenda-se o uso de outros algoritmos de aprendizagem de máquina tanto supervisionado como não supervisionado para aferir se a classificação possui melhor desempenho.

Observando o comportamento da máquina virtual em que são submetidos os *ransomwares* para análise dinâmica pode-se verificar pontuais alterações, como por exemplo: (i) aumento no nível de processamento; e (ii) lentidão do sistema operacional Essas alterações podem indicar alteração significativa de estado de *hardware*, como por exemplo: aumento de temperatura do processador, alta reserva de endereço de memória e alteração na rotação de *cooler* do processador. Tais alterações podem indicar que a

ação de um *ransomware* altera o comportamento físico de recursos computacionais que podem ser utilizados para auxiliar ou aprimorar a classificação desse *malware.* Tais alterações não podem ser comprovadas neste estudo pois este trabalho utiliza máquinas virtuais para coleta.

Propõem-se ainda, aplicar a proposta deste trabalho para classificação dos *malwares* existentes, como por exemplo: vírus, *worm*, *botnet, spyware, backdoor,* cavalo de tróia e *rootkit.*

Por fim, utilizar esta pesquisa em redes de sensores e Internet das Coisa – Internet of Things (IoT) aferindo se é possível detectar a presença de *ransomware* nesse tipo de tecnologia e a performance de detecção.

Ressalte-se que este trabalho baseou-se em coletar, analisar e extrair atributos de *ransomwares* para sistemas operacionais Microsoft Windows. Os atributos extraídos para detecção e classificação podem não ser suficientes, não apresentar resultados semelhantes ou divergentes em outros sistemas operacionais, como GnuLinux Debian, macOS e/ou Solaris. E nenhum dos arquivos utilizados para construção do vetor contendo os atributos de *ransomware* e benignos são do tipo que afetam arquivos de banco de dados.

REFERÊNCIAS

ANDRONIO, N. **Heldroid : Fast and Efficient Linguistic-Based Ransomware Detection**. Chicago, 2015. 85 f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Ciências da Computação, Universidade de Illinois, Chicago, 2015.

BRANCO, R. R.; BARBOSA, G. N.; NETO, P. D. **Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies**. Em: Black Hat Technical Security Conf.(Las Vegas, Nevada, 2012).

CALDAS, D. M. **Análise e Extração de Características Estruturais e Comportamentais de *Malware.*** Brasília, 2016. 105 f. Dissertação (Mestrado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, 2016.

CARVALHO, A. C. P. L. F.; GAMA, J.; LORENA, A. C.; FACELI, K. **Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina.** Edição 1.São Paulo: LTC, 2011. 394 p.

CERT.BR. **Cartilha de Segurança para Internet**. Centro de Estudos e Respostas e Tratamento de Incidentes de Segurança no Brasil, Brasil, maio 2017. Disponível em: <https://cartilha.cert.br/ransomware. Acesso em: 20 set. 2017.

COHEN, F: A Coefficient of agreement for nomial scales, educacional and psychological measurement. **Education and Psychological Measurement**, Nova York, p. 37-46, 1960

GORMAN, G. O.; MCDONALD, G. Ransomware : A Growing Menace. **Symantec**, v. 1, p. 16, 2012.

HOSFELT, D. D. **Automated detection and classification of cryptographic algorithms in binary programs through machine learning by Declaration of Authorship**. Baltimore, 2015. 33 f. Dissertação (Mestrado em Engenharia da Computação) – Departamento de Ciências da Computação, Universidade Johns Hopkins, Baltimore, 2015.

KASPERSKY, **2016: *a year of ransomware in spam,*** 2017. Disponível em: <https://www.kaspersky.com/about/press-releases/2017_2016-a-year-of-ransomware-in-spam> Acesso em: 20 set. 2017.

LANDIS, J. R.; KOCH, G. G.:The measurement of observer agreement for categorical data. **International Biometrics Society,** v.33, n.1 p.159-174, Mar 1977

LUGER, GEORGE F. **Inteligencia Artificial.** 6ª ed. [S.I.]: Editora Pearson Education, 2008. 632p.

LUTZ, N. **Towards Revealing Attackers´ Intent by Automatically Decrypting Network Traffic.** Suiça, 2008. 52 f. Dissertação (Mestrado em Engenharia da Computação) Instituto Federal de Tecnologia de Zurique, Suiça, 2008.

LYDA, R.; HAMROCK, J.: **Using entropy analysis to find encrypted and packed malware**. IEEE Security & Privacy 5, 40–45, 2007.

MALIN, C. H; CASEY, E; AQUILINA, J. M. **Malware Forensics Field Guide For Windows Systems. Elsevier.** 1ed. Waltham: Editora Elsevier Inc, 2012. 517p.

MITCHELL, T. M. **Machine Learning**. 1 ed. Edmond: McGraw-Hill Science/Engineering, 1997. 432p.

MONARD, M. C.; BARANAUSKAS, J. A. **Sistemas Inteligentes. Fundamentos e Aplicações**. 1.ed. Barueri: Editora Manole Ltda,, 2003. 527p.

NATH, H. V.; MEHTRE, B. M. **Static Malware Analysis Using Machine Learning Methods.** School of Computer and Information Sciences (SCIS), University of Hyderabad, India, 2014.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.: Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v.12, p. 2825–2830, Out. 2011.

RIEHLE, D. **Framework Design:** A Role Modeling Approach**.** 2000. 229 f. Tese (Doutorado) - SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, Hamburgo, 2000.

SANTOS, I.; BREZO, F.; UGARTE-PEDRERO, X.; BRINGAS, P. G. Opcode sequences as representation of executables for data-mining-based unknown malware detection. **Information Sciences**, n 231, p 64-82, Aug. 2013.

SCIKIT-LEARN. Scikit-learn user guide. Release 0.19.1. 2017. Disponível em: <http://scikit-learn.org/stable/_downloads/scikit-learn-docs.pdf>. 23 Out. 2017. Acesso em: 10 nov. 2017.

SCHULTZ, M.; ESKIN, E.; ZADOK, F.; STOLFO, S. **Data Mining Methods for Detection of New Malicious Executables**, Em: Proceedings of the 22nd IEEE Symposium on Security and Privacy, p. 38–49, 2001.

SCHMIDT, J. L. S.; DODONOV, E.; GUARDIA, E. **Utilizando a análise dinâmica para entender códigos maliciosos: Um estudo de caso.** São Carlos: Tecnologia, Infraestrutura e Software. v.3, n.1, p. 102-113, Abr 2014.

SGANDURRA, D.; GONZÁLEZ, L. M.; MOHSEN, R.; LUPU, E. C. **Automated Dynamic Analysis of Ransomware : Benefits, Limitations and use for Detection.** Cornel University Library. 12p. Sept 2016.

SHAFIQ, M.; TABISH, S.; FAROOQ, M. **Pe-probe: leveraging packer detection and structural information to detect malicious portable executables.** Em: Proceedings of the Virus Bulletin Conference, pp. 29–33, 2009.

SIKORSKI, M. e HONIG, A. **Pratical Malware Analysis. The Hands-On Guide to Dissecting Malicious Software**. San Francisco: No Starch Press Inc, 2012. 450p.

SOPHOS, **2018 Malware Forecast: ransomware hits hard, continues to evolve,** 2017. Disponível em: <https://news.sophos.com/en-us/2017/11/02/2018-malware-forecast-ransomware-hits-hard-crosses-platforms>. Acesso em: 10 mar. 2018.

SUMMERFIELD, M. Programming in Python 3: a complete introduction to the Python language.2 ed. [S.l.]:Addison-Wesley Professional, 2009. 648p.

APÊNDICE A

Quadro 1: Relação de APIs usadas por malware.

AdjustTokenPrivileges	AttachThreadInput	BitBlt
CallNextHookEx	CertOpenSystemStore	CheckRemoteDebuggerPresent
CoCreateInstance	ConnectNamedPipe	ControlService
CreateFile	CreateFileMapping	CreateMutex
CreateProcess	CreateRemoteThread	CreateService
CreateThread	CryptAcquireContext	CreateToolhelp32Snapshot
DeleteFile	DeviceIoControl	DllCanUnloadNow
DllGetClassObject	DllInstall	NtQueryInformationProcess
DllUnregisterServer	EnumProcesses	EnableExecuteProtectionSupport
DllRegisterServer	FindFirstFile	EnumProcessModules
FindNextFile	FindResource	FindWindow
FtpPutFile	GetAdaptersInfo	GetAsyncKeyState
GetDC	GetForegroundWindow	GetModuleFilename
GetKeyState	GetModuleHandle	GetProcAddress
GetStartupInfo	GetSystemDefaultLangId	GetTempPath
GetThreadContext	GetTickCount	GetVersionEx
GetWindowsDirectory	InternetOpen	InternetOpenUrl
InternetReadFile	InternetWriteFile	IsDebuggerPresent
IsNTAdmin	IsWoW64Process	LdrLoadDll
LoadLibrary	LoadResource	LockResource
MapViewOfFile	MapVirtualKey	LsaEnumerateLogonSessions
Module32Next	Module32First	MmGetSystemRoutineAddress
NetScheduleJobAdd	NetShareEnum	NtQueryDirectoryFile
OpenMutex	OpenFile	NtSetInformationProcess
OleInitialize	OpenSCManager	NtQueryInformationThread
OpenProcess	OutputDebugString	Process32Next
PeekNamedPipe	Process32First	QueueUserAPC
QueryPerformanceCounter	ReadFile	RegisterClassExW
ReadProcessMemory	RegCreateKeyEx	RegDeleteKey
RegEnumKeyEx	RegEnumValue	RegisterHotKey
RegOpenKey	ResumeThread	RegisterServiceCtrlHandler
RtlCreateRegistryKey	RtlWriteRegistryValue	SamIConnect
SamIGetPrivateData	SamQueryInformationUse	SetFileTime
SetThreadContext	SetWindowsHookEx	SetWindowTextW

WinExec	ShellExecute	SfcTerminateWatcherThread
ShowWindow	SuspendThread	StartServiceCtrlDispatcher
TerminateProcess	Thread32First	Thread32Next
ToolhelpRead	ProcessMemory	URLDownloadToFile
VirtualAllocEx	VirtualProtectEx	WideCharToMultiByteº
WlxLoggedOnSAS	WriteProcessMemory	Wow64Disable
WriteFile	WSAStartup	Wow64FsRedirection

Fonte: Adptado de Sikorski (2012)