MINISTÉRIO DA DEFESA

EXÉRCITO BRASILEIRO

DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA

INSTITUTO MILITAR DE ENGENHARIA

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

KIZZY FERNANDA TERRA FERREIRA DOS REIS

LETÍCIA CREMASCO

IDENTIFICAÇÃO DE MALWARES UTILIZANDO ALGORITMOS DE APRENDIZADO DE MÁQUINA

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA

(Real Academia de Artilharia, Fortificação e Desenho/1792)

KIZZY FERNANDA TERRA FERREIRA DOS REIS

LETÍCIA CREMASCO

Trabalho de Iniciação à Pesquisa apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia do Instituto Militar de Engenharia, como requisito parcial para conclusão do curso.

Orientadores: Prof. Julio Cesar Duarte – D.C.

Prof. Claudio Gomes de Mello – D.C.

Rio de Janeiro

2013

INSTITUTO MILITAR DE ENGENHARIA

KIZZY FERNANDA TERRA FERREIRA DOS REIS

LETÍCIA CREMASCO

IDENTIFICAÇÃO DE MALWARES UTILIZANDO ALGORITMOS DE APRENDIZADO DE MÁQUINA

Trabalho de Iniciação à Pesquisa apresentado ao Curso de Engenharia de Computação do Instituto Militar de Engenharia – IME - como requisito parcial para conclusão do curso.

Orientadores: Maj QEM Julio Cesar Duarte – D.C.

Ten Cel QEM Claudio Gomes de Mello – D.C.

Aprovada em 2013 pela seguinte Banca Examinadora:

_____

Prof. Julio Cesar Duarte – D.C.do IME - Presidente

_____

Prof. Claudio Gomes de Mello – D.C. do IME

_____

Profª. Raquel Coelho Gomes Pinto – D.C do IME

_____

Prof. Ricardo Choren Noya – D.C.do IME

Rio de Janeiro

2013

SUMÁRIO

LISTA DE ILUSTRAÇÕES 6

LISTA DE TABELAS 7

1 INTRODUÇÃO 10

1.1 Motivação 10

1.2 Objetivo 11

1.3 Justificativa 11

1.4 Metodologia 12

1.5 Estrutura da Monografia 12

2 ANÁLISE DE MALWARE 13

2.1 Análise Estática 13

2.1.1 O Formato *Portable Executable* (PE) 14

2.1.1.1 Cabeçalho MZ do DOS 15

2.1.1.2 Fragmento (*stub*) do DOS 15

2.1.1.3 Cabeçalho de Arquivo PE 16

2.1.1.4 Cabeçalho Opcional 16

2.1.1.5 Cabeçalho das Seções 18

2.1.1.6 Seções 18

2.1.1.7 Análise dos Principais Atributos do Formato PE 19

2.1.2 O formato ELF 20

2.1.3 Outros formatos de arquivo binário executável 21

2.1.3.1 O formato a.out 21

2.1.3.2 O formato Mach-O 21

2.1.3.3 O formato DOS MZ executable 22

2.1.3.4 O formato PEF 22

2.2 Análise Dinâmica 22

3 APRENDIZADO DE MÁQUINA 25

3.1 Árvore de Decisão 28

3.1.1 ID3 31

3.1.2 Random Forest 31

3.2 SVM 32

3.2.1 SVMs não-lineares 33

3.3 Validação Cruzada 34

3.4 FAMA 34

4 CLASSIFICAÇÃO AUTOMÁTICA DE MALWARE37

4.1 Conformação dos dados37

4.2 Processamento 40

4.3 Validação 41

4.3.1 Experimento 01 43

4.3.2 Experimento 02 44

4.3.3 Experimento 03 44

4.3.4 Análise comparativa dos experimentos.....45

5 CONCLUSÃO 46

6 REFERÊNCIAS BIBLIOGRÁFICAS 47

LISTA DE ILUSTRAÇÕES

Figura 2.1 Estrutura do formato do *Portable Executable* 15
Figura 3.1 Árvore de Decisão para problema de espera para um jantar em um restaurante..30
Figura 3.2 Diagrama de classes do FAMA.....35
Figura 4.1 Relatório gerado pelo Cuckoo SandBox para o arquivo 7zG.exe.....39

LISTA DE TABELAS

Tabela 2.1 Cabeçalho de Arquivo PE.....16
Tabela 2.2 Cabeçalho Opcional.....17
Tabela 2.3 Cabeçalho das Seções.....18
Tabela 2.4 Atributos Principais de um PE (YONTS, 2012).....19
Tabela 4.1 Análise estática do *7zG.exe*.....37
Tabela 4.2 Tabela de dados.....38
Tabela 4.3 Tabela de pré-processamento – Análise dinâmica.....38
Tabela 4.4 Tabela dos parâmetros do SVM..... 41
Tabela 4.5 Tabela de resultados do teste com SVM.....42
Tabela 4.6 Tabela de resultados do Random Forest.....43
Tabela 4.8 Tabela de melhores resultados de cada experimento.....45

RESUMO

Com o crescimento das ameaças virtuais, faz-se necessário criar um conhecimento avançado sobre o funcionamento dos *malwares* de tal forma a permitir um melhor entendimento de suas técnicas de replicação e infecção. Nesse contexto, insere-se a importância da identificação de arquivos maliciosos. Para tanto, podem ser utilizados algoritmos de aprendizado de máquina, dispensando a necessidade da aplicação de um especialista na tarefa.

Por essa razão, este trabalho teve por objetivo realizar uma pesquisa sobre algoritmos de aprendizado de máquina utilizados na classificação de *malware*. Como objetivo secundário, foram realizados experimentos de classificação de *malware* utilizando instanciações do framework FAMA com diferentes algoritmos de aprendizado de máquina, a fim de verificar e comparar o desempenho desses classificadores.

A partir da utilização dos algoritmos de classificação SVM e ID3 em instâncias do framework FAMA, constatou-se que embora a utilização de dados da análise estática para classificar códigos binários como maliciosos apresente bons resultados, o uso dos dados da análise dinâmica para realizar a classificação possui uma melhor acurácia.

A taxa de acurácia obtida para os exeprimentos em que foram utilizados apenas os dados gerados pela análise estática foi entre 75,92% e 76,08%, ao passo que para os dados obtidos na análise dinâmica a acurácia ficou entre 72,79% e 82,32%. Outro fator relevante foi que em todos os experimentos com o conjunto de dados da análise dinâmica a acurácia obtida ficou acima de 76,06%, representando uma melhora significativa em relação aos resultados obtidos com os dados referentes à análise estática.

ABSTRACT

The increasing of cyber threats requires developing an advanced knowledge about malwares behavior in order to enable a better understanding of their replication and infection techniques. In this context, the identification of malicious files becomes very important and can be done using machine learning algorithms that eliminate the need of a human to acquire this knowledge.

Hence, this research aims to study machine learning algorithms which might be used to classify malwares. As a secondary goal we will make malware classification experiments using different extensions of the framework of machine learning FAMA in order to verify and compare the obtained classifiers.

After use the machine learning algorithms SVM and ID3 in extensions of framework FAMA we can realize that the experiments which used the data generated by the dynamic analysis produces better results than the data generated by the static analysis.

The accuracy rate obtained in the experiments of malware classification using the static analysis data was between 75.92% and 76.08%, whereas for the dynamic analysis data it was between 72.79% and 82.32%. Another interesting point was that all the experiments using dynamic analysis data had accuracies rates above 76.06% this represents a great improvement in relation to the experiments using static analysis data.

1 INTRODUÇÃO

O objetivo principal do aprendizado de máquina é a geração de sistemas computacionais que possam substituir o trabalho humano a um baixo custo, com desempenho compatível, diminuindo, assim, a quantidade de recursos alocados para uma determinada tarefa. Também é desejável que tais sistemas possam aprender com a experiência, melhorando continuamente o seu desempenho. Existem diversas aplicações para aprendizado de máquina, tais como: processamento de linguagem natural, mecanismos de busca para internet, detecção de fraudes, reconhecimento de padrões, entre outras.

Por outro lado, a análise de código malicioso tem por objetivo alcançar o entendimento do funcionamento de um *malware* em um sistema operacional.

A detecção de *malware* é tipicamente um problema de classificação e agrupamento, pois sua finalidade é distinguir um programa malicioso de um não malicioso, ou ainda distinguí-lo segundo sua finalidade ou objetivo. Por essa razão, pode-se utilizar o aprendizado de máquina supervisionado. É neste contexto que este trabalho de pesquisa se insere tendo como principal enfoque o estudo sobre aplicações de análise de *malware* que utilizam algoritmos de aprendizado de máquina.

1.1 Motivação

No ano de 2010, cerca de 67 milhões de programas e arquivos nocivos foram criados entre os meses de janeiro e outubro, correspondendo a uma média de 220 mil novos vírus e *malwares* criados por dia segundo a empresa Panda. Também, segundo a companhia, foram cadastradas no *Collective Intelligence* (um sistema que mensura a segurança na internet), uma quantidade de ameaças que ultrapassou a marca de 200 milhões. (RAMOS, 2011)

A análise de *malware* insere-se nesse contexto como uma tarefa básica que possibilita a criação de assinaturas para estes códigos maliciosos a fim de permitir uma futura verificação

de seus padrões e de variantes. Entretanto, esse tipo de análise é frequentemente realizado manualmente e necessita de um especialista na área que domine a estrutura dos arquivos maliciosos e seja capaz de analisá-los criteriosamente.

As técnicas de aprendizado de máquina, por sua vez, possibilitam que a análise dos arquivos maliciosos seja feita sem exigir um especialista no assunto. Nesses casos, um classificador derivado de um algoritmo de aprendizado tenta “reproduzir” o comportamento do especialista, realizando a tarefa, em um tempo bem menor e com um desempenho satisfatório para o problema em questão.

1.2 Objetivo

Este trabalho tem por objetivo realizar uma pesquisa sobre algoritmos de aprendizado de máquina que podem ser utilizados na classificação de *malware*.

Como objetivo secundário, serão realizados experimentos de classificação de *malware* utilizando instanciações do FAMA com diferentes algoritmos de aprendizado de máquina, a fim de verificar e comparar o desempenho desses classificadores.

1.3 Justificativa

Com a crescente importância da Defesa Cibernética, torna-se cada vez mais imprescindível a obtenção de um sistema de detecção de códigos maliciosos, com o intuito de evitar possíveis ataques aos ativos. É nesse âmbito que o desenvolvimento de estudos na análise de *malware* se destaca.

O lançamento do primeiro sistema nacional de monitoramento contra ataques vindos da Internet pelo Centro de Defesa Cibernética (CDCiber) é um exemplo do quão relevante é a análise de códigos maliciosos. Esse sistema além de ter sido utilizado durante a Rio +20, deverá ser utilizado na Copa do Mundo de 2014 e nas Olimpíadas de 2016 com objetivo de garantir a segurança de informação para seus participantes (Kleina,2012).

Além disso, a automação de sistemas torna indústrias e empresas mais vulneráveis à ataques cibernéticos, o que justifica o desenvolvimento desse trabalho em análise de malware.

1.4 Metodologia

Este trabalho está estruturado em duas etapas principais: a primeira etapa da pesquisa compreende o estudo sobre análise estática e dinâmica de *softwares* maliciosos, bem como sobre algoritmos de aprendizado de máquina; a segunda etapa compreende os testes de desempenho dos algoritmos estudados. Para executarmos esta última etapa, será necessário, primeiramente, a implementação de uma classe de processamento dos dados adquiridos a partir da análise de malware; feito isso, serão iniciados os testes de desempenho com os algoritmos de aprendizado de máquina candidatos a serem utilizados na classificação de *malware*.

1.5 Estrutura da Monografia

O capítulo 2 trata da análise de *softwares* maliciosos, que pode ser estática ou dinâmica. Nesse capítulo, descreve-se a estrutura de um arquivo binário do formato *Portable Executable*, o qual é alvo frequente de ameaças virtuais por ser o formato desenvolvido pela Microsoft para sistemas operacionais Windows.

No capítulo 3, são apresentados conceitos sobre aprendizado de máquina (AM) supervisionado e não supervisionado. Além disso, é realizada a descrição de alguns algoritmos de AM que são utilizados na tarefa de classificação de *malwares*, bem como é apresentado o *framework* FAMA.

No capítulo 4, estão descritos os experimentos de classificação de *malware* realizados, bem como as etapas de processamento dos dados que antecederam o experimento. Além disso, no final do capítulo são apresentados os resultados obtidos e as comparações entre os classificadores instanciados.

Finalmente, no capítulo 6, é feita uma conclusão, expondo os pontos mais importantes da pesquisa e apresentando sugestões para trabalhos futuros.

2 ANÁLISE DE MALWARE

A facilidade de utilização dos diversos serviços eletrônicos disponíveis nos mais variados *websites*, como *e-mail*, *games*, *chats*, impulsionada, principalmente, pelo processo de democratização do acesso às tecnologias de informação, incentiva o crescimento das ameaças dos agentes de *softwares* maliciosos como os vírus, cavalos de tróia e *worms*, também chamados de *malwares* ou *malicious software*.

Milhares de arquivos maliciosos mais potentes são desenvolvidos a cada dia, ao passo que os antivírus e os dispositivos de defesa estão se tornando ineficazes. Além disso, a falta de padronização e documentação da classificação dos *malwares* já identificados dificulta a melhoria na defesa dos sistemas. Torna-se, portanto, muito importante desenvolver conhecimentos avançados sobre o comportamento desses agentes para que se possa evitar sua ação e possíveis danos.

A análise de programas maliciosos tem como principal objetivo monitorar o funcionamento destes em um sistema operacional, verificando o tipo de informação que é captada, se há operações na rede, entre outras ações. Essa análise será classificada como estática, se for realizada antes da execução do arquivo; ou dinâmica, se for realizada com o arquivo em execução, nesse caso, o monitoramento do *malware* é feito em um ambiente controlado, tal como uma máquina virtual.

2.1 Análise Estática

A análise estática de *malware* permite a compreensão das funcionalidades básicas de um arquivo binário malicioso e é realizada antes da execução do arquivo. Contudo, para que seja possível realizá-la, é necessário dominar as peculiaridades do formato de cada arquivo suspeito que esteja sendo estudado para que se possa, então, extrair e analisar todas as informações que indicam se o arquivo é realmente malicioso.

Existem vários formatos de arquivos binários executáveis, nas seções 2.1.2 e 2.1.3 que seguem são apresentadas as características principais de alguns desses formatos, podem ser destacados, entretanto, dois formatos principais: *Portable Executable* (PE) que é o formato padronizado pela *Microsoft* para sistemas operacionais *Windows* e o formato ELF que é o padrão de arquivo binário para os sistemas *Unix.*

Nesta pesquisa, escolheu-se como objeto de análise, os arquivos binários do formato PE, uma vez que este é o formato compatível com sistemas operacionais *Windows* os quais são alvos mais frequentes de ameaças virtuais, já que a maioria dos *worms* é desenvolvida para esta plataforma. Desta forma, a análise de arquivos desse formato abrange a maioria dos executáveis maliciosos e permite que sejam feitas classificações que realmente representem de forma fidedigna o universo dos arquivos binários maliciosos e não maliciosos.

2.1.1 O Formato *Portable Executable* (PE)

O formato de arquivo *Portable Executable* (PE) foi projetado pela *Microsoft* e padronizado, em 1993, pelo Comitê do *Tool Interface Standart* (TIS), que é constituído pela *Microsoft, Intel, Borland, Watcom, IBM* e outras empresas. Este formato foi baseado no *Common Object File Format* (COFF), o qual havia sido usado originalmente para arquivos de objetos e executáveis nos vários sistemas *UNIX* e no *VMS*. Além disso, a sua denominação deve-se a portabilidade dos arquivos desse tipo, que possibilitam a implementação em diversas plataformas, *x86, MIPS®, Alpha*, entre outras, sem que seja necessário reescrever as ferramentas para desenvolvimento. (DUMMER, DANIEL, 2006)

Os arquivos obedecem a um padrão de armazenamento de dados, em que estão encapsuladas todas as informações necessárias ao carregamento, também chamado de *loader,* do sistema operacional. Essas informações são a representação binária das instruções de máquina para o processador.

O formato PE é constituído pelas seguintes estruturas de dados: cabeçalho MZ do DOS, fragmento, ou *stub*, do DOS, cabeçalho de arquivo PE, cabeçalho de imagem opcional, tabela de seções, a qual possui uma lista de cabeçalhos de seção, diretórios de dados, o qual contém os ponteiros para as seções, e por último as seções propriamente ditas conforme a Figura. 2.1.

A análise de atributos do formato PE permite identificar arquivos binários suspeitos e prognosticar danos. Os atributos principais de cada executável são analisados e quando se enquadram em determinadas condições o arquivo pode ser considerado suspeito. Por exemplo, um executável é suspeito se o seu atributo *checksum* não corresponde a um valor válido, isto é, o valor deste atributo é diferente do valor de *checksum* estimado através de cálculos. Adiante, nesta seção, discutiremos outras possíveis condições que indicam valores suspeitos dos atributos.

Figura. 2.1 Estrutura do formato *Portable Executable* (DUMMER, 2006)

2.1.1.1 Cabeçalho MZ do DOS

Os primeiros 64 *bytes* de qualquer arquivo PE constituem o cabeçalho do DOS. Os primeiros dois *bytes* deste cabeçalho constituem a assinatura do DOS. A primeira palavra (conjunto de dois *bytes*) da assinatura sempre é a sequência “MZ”, ou seja, “4D5A” em hexadecimal, por essa razão através dessa assinatura é possível fazer a validação do cabeçalho DOS. O cabeçalho DOS armazena outra informação importante do arquivo executável: seus últimos quatro *bytes* indicam o *offset* do cabeçalho de arquivo PE.

2.1.1.2 Fragmento (*stub*) do DOS

Para o formato PE, o fragmento do DOS é um executável compatível com o MS-DOS 2.0, constituído por aproximadamente 100 *bytes*, cuja função é exibir mensagens de erro. Uma parte dos valores tem o correspondente em ASCII de *"This program cannot be run in DOS mode"*, que é a *string* mostrada caso se tente executar este programa a partir do DOS.

2.1.1.3 Cabeçalho de Arquivo PE

Signature	Assinatura do cabeçalho PE
Machine	Tipo de máquina previsto para rodar o executável
NumberOfSections	Número de seções
TimeDateStamp	Carimbo de data e hora
PointerToSymbolTable	Ponteiro para tabela de símbolos e número de símbolos
SizeOfOptionalHeader	Tamanho do cabeçalho opcional
Characteristics	Características

Tabela 2.1 Cabeçalho de Arquivo PE

O cabeçalho do arquivo ocupa 24 *bytes* e seus componentes encontram-se descritos na Tabela 2.1 acima.

Signature é a assinatura do cabeçalho PE indicando o início deste cabeçalho e sempre é a sequência “PE” seguida de dois zeros, correspondente à palavra dupla “50450000” em hexadecimal. Outro componente importante do cabeçalho do arquivo PE é o *NumberOfSections* que indica o número de seções após o cabeçalho.

2.1.1.4 Cabeçalho Opcional

Imediatamente após o cabeçalho do arquivo vem o cabeçalho opcional que contém informações de como o arquivo PE deve ser tratado. Os componentes desse cabeçalho encontram-se na Tabela 2.2 a seguir.

Magic	Identifica o estado do *Image File*
MajorLinkerVersion,	Número de versão
MinorLinkerVersion	Número de versão
SizeOfCode	Tamanho do código de seção
SizeOfInitializedData	Tamanho da seção de dados inicializada
SizeOfUninitializedData	Tamanho da seção de dados não inicializada
AdressOfEntryPoint	Endereço do *Entry Point*
BaseOfCode	Endereço relativo ao *ImageBase* do início da seção CODE
BaseOfData	Endereço relativo ao *ImageBase* do início da seção DATA
ImageBase	Endereço do primeiro byte da imagem quando carregada na memória
SectionAlignment	Alinhamento (em bytes) das seções quando carregadas na memória
FileAlignment	Fator de alinhamento usado para alinha o *RawData* das seções no *Image File*
MajorOperatingSystemVersion	Número de versão requerido pelo sistema operacional.
MinorOperatingSystemVersion	Número de versão requerido pelo sistema operacional.
MajorSubsystemVersion	Número de versão do subsistema
MinorSubsystemVersion	Número de versão do subsistema
SizeOfImage	Tamanho do *Image*(em bytes) incluindo todos os cabeçalhos.
SizeOfHeaders	Múltiplo do *FileAlignment*
CheckSum	*Image File* checksum
Subsystem	Subsistema requerido para executar esta *Image*
NumberOfRvaAndSizes	Descreve localizações e tamanhos no cabeçalho opcional.
DataDirectory	Diretórios de dados

Tabela 2.2 Cabeçalho Opcional

Alguns dos componentes mais importantes deste cabeçalho são o *AdressOfEntryPoint*, que indica a posição relativa de memória (RVA- *Relative Virtual Adress*) do ponto de entrada do código do executável; e o *DataDirectory* armazenado nos 128 últimos bytes do cabeçalho opcional.

O *DataDirectory* compreende 16 diretórios de dados que descrevem a posição relativa de memória (um RVA de 32 bits) e o tamanho (também de 32 bits, chamado *Size*) de cada uma das diferentes seções que seguem as entradas de diretório.

2.1.1.5 Cabeçalho das Seções

Imediatamente após o cabeçalho do arquivo PE, está o cabeçalho das seções. Este cabeçalho é um array de estruturas, que armazena informações de cada uma das seções. Cada estrutura tem 40 bytes, e contém as informações descritas na Tabela 2.3 abaixo.

Name1	Um *array* de nomes das seções
VirtualSize	Tamanho total da seção quando carregada na memória
SizeOfRawData	Tamanho dos dados inicializados no disco
PointerToRawData	Múltiplo do *FileAlignment*
Characteristics	*Flag* descrevendo as características das seções

Tabela 2.3 Cabeçalho das Seções

2.1.1.6 Seções

Após os cabeçalhos das seções seguem as seções propriamente ditas. Dentro do arquivo, elas estão alinhadas em *FileAlignment* bytes, ou seja, após o cabeçalho opcional e após cada uma das seções haverá bytes zerados de preenchimento para que seja atingido o tamanho *FileAlignment*. Por outro lado, quando carregadas na memória RAM, as seções ficam

alinhadas segundo o tamanho designado pelo *SectionAlignment.* Além disto, as seções estão ordenadas pelos seus RVA. Também é possível encontrar o início das seções através do *PointerToRawData* (ponteiro de dados) ou através do *VirtualAddress* de maneira que a informação dos alinhamentos passa a ter menos relevância.

É interessante notar que existe um cabeçalho de seção para cada seção e cada diretório de dados apontará para uma das seções. Entretanto, vários diretórios de dados podem apontar para uma mesma seção e podem existir seções que não sejam apontadas pelo diretório de dados.

2.1.1.7 Análise dos Principais Atributos do Formato PE

Entre os atributos descritos acima, alguns são determinantes na análise dos arquivos binários para que se possa classificá-los como maliciosos. Na Tabela 2.4 a seguir são apresentados os atributos considerados principais segundo (LIGH, 2011)**,** bem como as condições que determinam um comportamento suspeito do arquivo para cada um dos atributos.

Atributos	**Condições**
Ano	Menor que 1992 ou maior que 2012
NumberOfSections	Menor que 1 ou maior que 9
AdressOfEntryPoint	Verifica se a seção do binário em que se encontra o EntryPoint é suspeita
CheckSum	Verifica se o checksum é válido
Version Info	***
APIs suspeitas (Import Directory)	Verifica se existem APIs suspeitas
NumberOfRvaAndSizes	Diferente de 16
Raw Size	Nulo
Virtual Size/Raw Size	Maior que 10
Section Entropy	Entre 0 e 1 ou maior que 7

Tabela 2.4 Atributos Principais de um PE (LIGH, 2011)

Por meio do uso das condições determinadas acima, podemos prever a classificação de um binário em código malicioso ou não. Porém, essa classificação não é absolutamente confiável, já que, dentre as restrições acima, a que possui maior taxa de detecção é a restrição

referente à Section Entropy, que é aproximadamente 23%, sendo a taxa de falsos positivos cerca de 2% (YONTS, 2012).

Para melhorar essa classificação, realiza-se a análise comportamental dos arquivos maliciosos em ambientes controlados, a fim de obter dados que unidos às condições acima, possibilitem a classificação de binários de forma confiável.

2.1.2 O formato ELF

O formato ELF *(Executable and Linkable Format)* foi escolhido em 1999 como padrão de arquivo binário para os sistemas *Unix*. Atualmente, o formato ELF já substituiu outros formatos de execução mais antigos, tais como a.out e COFF nos sistemas operacionais *Linux*, *Solaris*, IRIX, *FreeBSD*, *NetBSD, e OpenBSD*. Existem três formas principais de emprego do formato ELF: relocável, executável e objeto compartilhado. Os arquivos relocáveis são criados pelos compiladores e devem ser processados pelo *linker* antes de executar; os arquivos executáveis são arquivos que podem ser executados pelo sistema operacional; e os arquivos-objeto compartilhados são as bibliotecas dos sistemas.

Os arquivos do formato ELF podem ser interpretados de duas maneiras distintas, comumente denominadas *Linking View* (ou visão de ligação), em que os compiladores e *linkers* tratam o arquivo como um conjunto de seções; e *Execution View* (ou visão de execução), em que o *loader* trata o arquivo como um conjunto de segmentos. Um segmento normalmente contém várias seções. As seções serão processadas pelo *linker*, ao passo que os segmentos serão mapeados na memória pelo *loader.*

De maneira geral, o binário ELF apresenta um cabeçalho, que armazena informações gerais sobre o arquivo, tais como o tipo de arquivo (objeto compartilhado, biblioteca ou executável), arquitetura (MIPS, x86 68K etc.), versão, endereços de memória, etc; e uma área de dados que contém a tabela de cabeçalhos do programa, a qual armazena a informação do *offset* em que um segmento inicia e termina; a tabela de cabeçalho das seções, que especifica as seções do arquivo propriamente ditas (.text, .data ou .bss, por exemplo), bem como as seções dentro de cada segmento, quando for o caso. As seções armazenam bytes que podem ser códigos, dados ou comentários. Podem existir, entretanto, bytes que não pertencem a nenhuma seção, quando isso acontece esses bytes são intitulados bytes órfãos. A tabela de

cabeçalhos do programa é utilizada pelo sistema operacional, mais especificamente, para carregar o programa executável em memória.

2.1.3 Outros formatos de arquivo binário executável

Nas subseções anteriores foram descritos os formatos de executáveis .exe e .elf, a seguir destacamos sucintamente outros formatos conhecidos: a.out, Mach-O, DOS MZ e PEF.

2.1.3.1 O formato a.out

Este formato de arquivo era utilizado em versões dos sistemas operacionais *Unix-like* para carregar programas e em algumas versões antigas para o compartilhamento de bibliotecas.

Este nome era dado aos arquivos de saída assembler, entretanto continua sendo utilizado como padrão para arquivos de saída gerados por alguns compiladores, quando o nome do arquivo de saída não está especificado, mesmo que o arquivo não esteja no formato de um executável do tipo a.out.

2.1.3.2 O formato Mach-O

O formato Mach-O (Mach Object) é o formato padrão utilizado para armazenar programas e bibliotecas na MAC ABI (App Binary Interface). Um arquivo Mach-O contém três regiões principais: um cabeçalho que identifica o formato e contém outras informações básicas do arquivo, uma região com comandos *load* que definem as características de layout e ligação do arquivo; e uma região de segmentos, os quais podem conter zero ou mais seções.

O número exato de seções e *layout* de segmentos e seções está especificado na região que contém os comandos *load*.

2.1.3.3 O formato DOS MZ executable

O formato DOS MZ é o formato de arquivo executável utilizado para arquivos .exe em DOS. O arquivo pode ser identificado pela ASCII string "MZ" (hexadecimal: 5A 4D) no início do processo (o "número mágico"). "MZ" são as iniciais de Marcos Zbikowski, um dos desenvolvedores do MS-DOS.

O cabeçalho executável DOS contém informações de relocação, as quais permitem que vários segmentos sejam carregados em endereços de memória arbitrárias, e suporta executáveis maiores do que 64 KB, no entanto, o formato ainda requer limites de memória relativamente baixos. Estes limites podem ser contornados usando extensores do DOS.

2.1.3.4 O formato PEF

Os arquivos do formato PEF (Preferred Executable Format) também são chamados arquivos *Code Fragment Manager* (CFM). Este arquivo foi desenvolvido pela Apple Computer para uso em seu sistema operacinal Mac OS e foi otimizado para processadores RISC.

No Mac OS X, o formato de arquivo de Mach-O é o formato executável nativo. No entanto, o PEF ainda é suportado em computadores Macintosh baseados em PowerPC.

2.2 Análise Dinâmica

A análise dinâmica de *malware* caracteriza-se pela execução monitorada do arquivo binário suspeito em um ambiente controlado, geralmente uma máquina virtual.

O comportamento do arquivo executável compreende as atividades efetuadas por este no sistema operacional, tais como abrir um arquivo ou criar processos, considerando-se a ordem em que estas atividades são realizadas e a maneira como o sistema reage a essas atividades. Para monitorar os arquivos maliciosos é preciso executá-los em ferramentas especialmente projetadas para este fim e existem técnicas específicas para isso. Conforme a técnica utilizada a análise dinâmica pode ser classificada como manual ou automática.

Na análise manual, é possível observar como a entrada interage com o fluxo de execução e os dados do programa. Essa análise utiliza um depurador (*debugger*) que permite a observação de um programa durante sua execução ou um ambiente controlado como uma máquina virtual (VMWare, VirtualBox e outros) com programas de monitoramento. Um depurador possui tipicamente duas características básicas: habilidade de ajustar pontos de

parada (*breakpoints*) no programa e capacidade de verificar o estado atual do programa (registradores, memória, conteúdo da pilha). (DE ANDRADE,2013)

Por outro lado, na análise automática, o código malicioso é executado em um ambiente controlado que intercepta todas as alterações realizadas pelo *malware*, este ambiente é denominado *sandbox*. Esse tipo de ambiente permite que se faça o monitoramento dos arquivos binários maliciosos, minimizando os danos aos sistemas externos, uma vez que ao reiniciar o computador, as alterações provocadas pelo binário são apagadas do disco. A maioria dos *sandboxes* simula o sistema operacional Windows devido à grande quantidade de *malwares* escrita para este sistema.

Em geral, o *malware* é executado por quatro ou cinco minutos e, durante este tempo, são monitoradas de forma automática as ações pertinentes tanto ao *malware* quanto aos processos derivados dele. Após o período de monitoração, um relatório de atividades é gerado para análise. (DE ANDRADE,2013). Vale ressaltar que alguns *malwares* podem levar mais de cinco minutos para serem analisados, visto que alguns desses códigos maliciosos são implementados para que possam detectar que estão em um ambiente controlado e uma tendo identificado esse tipo de ambiente não executam nenhuma ação para que análise não o classifique como um binário suspeito, isto é, o *malware* tenta impedir sua detecção.

Entre os sandboxes mais conhecidos podemos destacar: GFISandbox (anteriormente chamado de CWSandbox) , Anubis , Norman Sandbox, ThreatExpert , Joebox, CaptureBat, Cuckoo Sandbox e Zero Wine.

No contexto desta pesquisa, a análise dinâmica é utilizada pra a verificação das API's utilizadas pelos agentes maliciosos durante sua execução. A partir de relatórios gerados pelo Cuckoo Sandbox é possível identificar quais APIs foram chamadas por determinado *malware* e quantas vezes esse binário malicioso chamou cada API. Essa informação, associada ao conteúdo extraído das referências bibliográficas e à análise estática dos principais atributos do executável, citados na seção 2.1.1, será utilizada para a classificação de arquivos suspeitos em maliciosos e não-maliciosos através de um algoritmo de aprendizado de máquina.

Primeiramente, é realizada uma coleta de dados que consiste na obtenção de conjuntos de códigos. Esse conjunto deve conter códigos maliciosos e não maliciosos. Em seguida, os comportamentos destes dados são identificados de forma automatizada, submetendo-os ao *Cuckoo Sandbox*. A partir desse processo, para cada dado coletado, relatórios de atividade no formato "*.csv*" são obtidos. (DE ANDRADE, 2013)

Logo após essa etapa, ocorre a análise comportamental customizada, a qual se constitui de duas etapas: pré-processamento dos dados, Aprendizado e Avaliação.

Durante a etapa de pré-processamento, os atributos mais importantes são identificados por meio de uma seleção sob todos os relatórios, sendo estes utilizados para a criação de um dicionário de termos. Em seguida, é verificada a frequência de cada termo do dicionário nos relatórios. Essa etapa será melhor detalhada no capítulo 4 (seção 4.1).

Na fase de aprendizado e avaliação, alguns algoritmos de classificação são utilizados e os desempenhos de cada um são comparados com os restantes. Além disso, o desempenho da metodologia adotada é analisado pela avaliação da acurácia, falsos positivos e falsos negativos.

3 APRENDIZADO DE MÁQUINA

O aprendizado de máquina é uma área da Inteligência Artificial que tem obtido grande importância atualmente, tendo aplicações no processamento de linguagem natural, diagnósticos médicos, bioinformática, reconhecimento de fala e de escrita, visão computacional entre outros.

Esse processo baseia-se na forma como os seres humanos e animais adquirem conhecimento. Assim, à medida que essa área se desenvolve, mais dados e informações a respeito de como se dá o conhecimento nesses seres são obtidos.

O aprendizado pode ser definido como qualquer mudança num sistema que melhore o seu desempenho na segunda vez que ele repetir a mesma tarefa, ou outra tarefa de mesma população. (SIMON, 1983)

O aprendizado consiste em examinar e experimentar as estratégias mais eficazes para a construção de programas que aprendem a partir da experiência, adquirindo conhecimento de forma automática. (MITCHELL,1997)

Apesar de parecer ser uma área nova, esta iniciou através da epistemologia no século 19. A epistemologia é um estudo que abrange a aplicação, a aquisição, a representação, a exposição e a explicação do conhecimento.

O processo de aquisição do conhecimento é um processo indutivo, pois tem como objetivo obter premissas a partir de casos particulares. Assim, esse processo parte de uma hipótese e obtém como solução, dentre um conjunto de dados, o dado que mais se aproxima dessa hipótese, e essa escolha é gerada a partir da interação com o ambiente.

É preciso definir, antes do processo, o que deve ser aprendido, como isto deve ser representado e qual método será utilizado para melhorar o desempenho do processo, o qual é chamado de método de realimentação.

Existem dois tipos de método de realimentação: o aprendizado supervisionado e o não supervisionado. Ambos são processos de obtenção de conhecimento de forma indutiva, porém, eles possuem diferenças que serão abordadas abaixo.

O aprendizado supervisionado gera regras de classificação e um conjunto de classes, utilizando um conjunto de dados rotulados, e a partir dessas regras, a máquina se torna capaz

de classificar uma informação qualquer em uma das classes anteriormente determinadas. (RUSSEL & NORVIG, 1995)

A obtenção do classificador pelo uso do aprendizado supervisionado é simples: basta aplicar o algoritmo sobre o conjunto de dados para que a partir deste sejam geradas as regras de classificação. É importante que esse conjunto de teste possua classificações diferentes, caso contrário, ao ser aplicado em uma informação, a classificação desta poderá ser errada, o que comprometerá o classificador produzido.

A grande vantagem deste processo é que o algoritmo e o classificador produzidos são independentes, logo, se ocorrer a falha mencionada anteriormente ou se desejar incluir novas classes, basta alterar o conjunto de dados de teste e realizar novos treinos para gerar um novo classificador.

O aprendizado não supervisionado é mais complexo do que o anterior. Ele consiste em extrair padrões de um conjunto de dados não rotulados. Esses padrões são obtidos pela busca de semelhanças entre os dados, e, em seguida, pelo agrupamento dos dados mais semelhantes em uma classe. Essas semelhanças são geradas a partir de modelos probabilísticos. (RUSSEL & NORVIG, 1995)

Esse processo possui como sub-rotina os processos de classificação e *clustering*, que é o processo de geração de novas classes. Aplica-se primeiro o segundo processo e, em seguida, classificam-se os dados. Se as classificações obtidas não estiverem de acordo com a tolerância estabelecida, retorna-se ao processo de *clustering*. Quanto mais completo seja o conjunto de dados, mais tempo o algoritmo levará para ajustar as semelhanças, porém a avaliação será mais confiável.

A Árvore de Decisão, a Regressão Logística, e SVM são exemplos de aprendizagem supervisionada, enquanto que a aprendizagem Bayesiana, aprendizagem de Hebbian são exemplos de aprendizagem não supervisionada. O foco do trabalho será em Árvore de decisão por este apresentar bons resultados na literatura em relação aos outros algoritmos de aprendizagem abordados (DE ANDRADE,2013). Além disso, há diversos algoritmos que utilizam o conceito de árvore de decisão como por exemplo, ID3, C4.5 e C5.0, os quais possuem muitas ferramentas que melhoram a precisão da classificação, aprimorando a mineração de dados e a poda da árvore, por exemplo. (QUINLAN, 2012)

O Naive Bayes e a Regressão Logística são dois exemplos de algoritmos de aprendizado de máquina. Estes algoritmos serão abordados abaixo.

O algoritmo de classificação de Naive Bayes assume que a presença ou a ausência de uma particular característica não está relacionada com a presença ou ausência de qualquer outra característica, dada na classe, ou seja, ele considera que cada uma das características contribui independentemente na probabilidade da classificação em determinada classe.

A condição de independência pode ser definida como:

Dadas as variáveis randômicas X, Y e Z, X é denominado condição independente de Y dado Z, se e somente se a probabilidade de distribuição que governa X é independente do valor de Y dado Z (MITCHELL,1997). Ou seja

$$(\forall i, j, k) P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k) \quad \textbf{(3.1)}$$

O algoritmo de classificação de Naive Bayes é baseado no teorema de Bayes que assume que os atributos X_1,..., X_n são todos condicionalmente independentes entre si, dado Y. Essa suposição simplifica drasticamente a representação de P(X|Y) e o problema de estimar isso, partindo dos dados de treinamento.

Quando X contem n atributos, os quais são condicionalmente independentes entre si, dado Y, temos:

$$P(X_i \dots X_n | Y) = \prod_{i=1}^{n} P(X_i | Y) \quad \textbf{(3.2)}$$

Para derivar o algoritmo de Naive Bayes, é necessário assumir que Y é uma variável discreta, e os atributos X_i,..., X_n são atributos discretos ou atributos reais. O objetivo é treinar um classificador que calculará a distribuição de probabilidade sobre os possíveis valores de Y, para cada nova instância X que se queira classificar. A expressão para a probabilidade em que Y irá assumir o k-ésimo valor possível, de acordo com Bayes, será:

$$P(Y = y_k | X_1 \dots X_n) = (P(Y = y_k) P(X_1 \dots X_n | Y = y_k)) / (\sum_j P(Y = y_j) P(X_1 \dots X_n | Y = y_j)) \quad \textbf{(3.3)}$$

Em que a soma é tomada por todos os valores y_j de Y. Assumindo que X_i são condicionalmente independentes, dado Y, pode-se usar a equação **(3.2)** para reescrever a equação **(3.3)**:

$$P(Y = y_k | X_1 \dots X_n) = (P(Y = y_k) \prod_i P(X_i | Y = y_k)) / (\sum_j P(Y = y_j) \prod_i P(X_i | Y = y_j)) \quad \textbf{(3.4)}$$

A equação **(3.3)** é uma equação fundamental para o classificador Naive Bayes. Dada uma nova instância X^{new} = <X_1... X_n>, essa equação mostra como calcular a probabilidade que Y irá

assumir para qualquer valor dado, qualquer valor atributo X^{new} dado e as distribuições P(Y) e $P(X_i|Y)$ estimados a partir dos dados de treinamento.

A Regressão Logística é uma aproximação de funções de aprendizado da forma $f: X \rightarrow Y$ ou P(Y|X) no caso em que Y é uma variável discreta, e X = $<X_1 ... X_n>$ é um vetor constituído de variáveis discretas ou continuas. Inicialmente, será considerado o caso em que Y é uma variável booleana, para simplificar a notação. Em seguida, os valores de Y serão extendidos para qualquer número finito de valores discretos.

Ela pode ser de dois tipos: binária ou multinomial. No caso binário, os valores são codificados como “0” ou “1”, e a variável alvo é referenciada com o valor “1”.

A Regressão Logística determina o impacto que múltiplas variáveis independentes apresentam simultaneamente na predição de uma variável dependente. Ela aplica a teoria de probabilidade binomial. Além disso, ela utiliza o método de maior probabilidade para encontrar a melhor função que se adapte ao problema. Esse método maximiza a probabilidade de classificar o dado observado na categoria apropriada, dado os coeficientes da regressão.

3.1 Árvore de Decisão

Árvore de Decisão é um modelo de classificação que é utilizado por diversos algoritmos de aprendizagem de máquina supervisionados. Este modelo utiliza a estratégia de dividir para conquistar: um problema complexo é decomposto em subproblemas mais simples e esta técnica é aplicada recursivamente a cada subproblema. (GAMA, 2004)

A árvore de decisão pode ser facilmente compreendida por ter uma simples representação. Nessa representação, cada nó de decisão da árvore corresponde a um teste para um determinado atributo, e seus ramos descendentes representam possíveis valores destes atributos. Além disso, cada folha pertence a uma classe e o caminho entre a folha e a raiz constitui uma regra de classificação. (GAMA, 2004)

O processo de partição baseia-se na escolha de um atributo do nó de decisão para que seja realizada a classificação. É importante salientar que o atributo teste para um nó corrente deve ser o que possui o maior ganho de informação, já que o processo de partição se dá a partir da inclusão de determinada informação a cada atributo teste. Um dos critérios mais utilizados para realizar a partição é a entropia.

A entropia é um critério que representa a impureza dos dados, já que esta é uma medida da falta de homogeneidade dos dados de entrada em relação à classificação dada pelo algoritmo. A entropia é máxima quando o conjunto de dados é heterogêneo, segundo (MITCHELL, 1997). A entropia pode ser enunciada como:

Dado um conjunto de entrada *S* que pode ter *c* classes distintas, a entropia será dada por:

$$Entropia(S) = \sum_{i=1}^{c} -p_i log_2^{p_i} \quad \textbf{(3.1.1)}$$

Em que p_i é a proporção de dados em *S* que pertencem à classe *i*. (MITCHELL, 1997)

Analisando a fórmula acima é possível perceber que a entropia máxima ocorre quando todas as classes do conjunto *S* possuem a mesma proporção, o que representa um conjunto de dados heterogêneo. Pela fórmula, esse valor é dado por log_2^c. Porém, se o conjunto de dados for homogêneo, todos os dados pertencerão à mesma classe, ou seja, $\exists i$ tal que $p_i = 1$, $\forall j \neq i \; p_j = 0$. Substituindo esses valores na fórmula, temos que a entropia mínima será 0.

O cálculo do ganho de informação de um atributo *A* de um conjunto de dados S pode ser feito simplesmente pelo cálculo da entropia do conjunto *S* menos a entropia do conjunto *A*. A entropia do atributo *A* pode ser facilmente obtida pela fórmula:

$$Entropia(A) = \sum_{x \in P(A)} \left(\frac{|S_x|}{|S|}\right) * Entropia(S_x) \quad \textbf{(3.1.2)}$$

Em que *P(A)* é o conjunto de valores que *A* pode assumir e S_x é um subconjunto de *S* formado pelos dados em que $A = x$, para todo *x* pertencente a *P(A)*. Essa entropia ocorre devido à partição de *S* em função do atributo *A*. (DA SILVA, 2005) Então, o ganho de informação é da forma:

$$ganho(S,A) = Entropia(S) - Entropia(A) \quad \textbf{(3.1.3)}$$

A construção de uma árvore de decisão tem como objetivo diminuir a entropia, ser consistente com o conjunto de dados e possuir o menor número possível de nós (DA SILVA, 2005).

Os algoritmos de classificação de Árvore de Decisão permitem que os usuários compreendam com facilidade as regras de classificação obtidas, já que essas regras consistem

em um caminho entre a raiz da árvore criada até uma folha desta. A figura abaixo representa uma Árvore de Decisão para o problema de espera para um jantar em um restaurante.

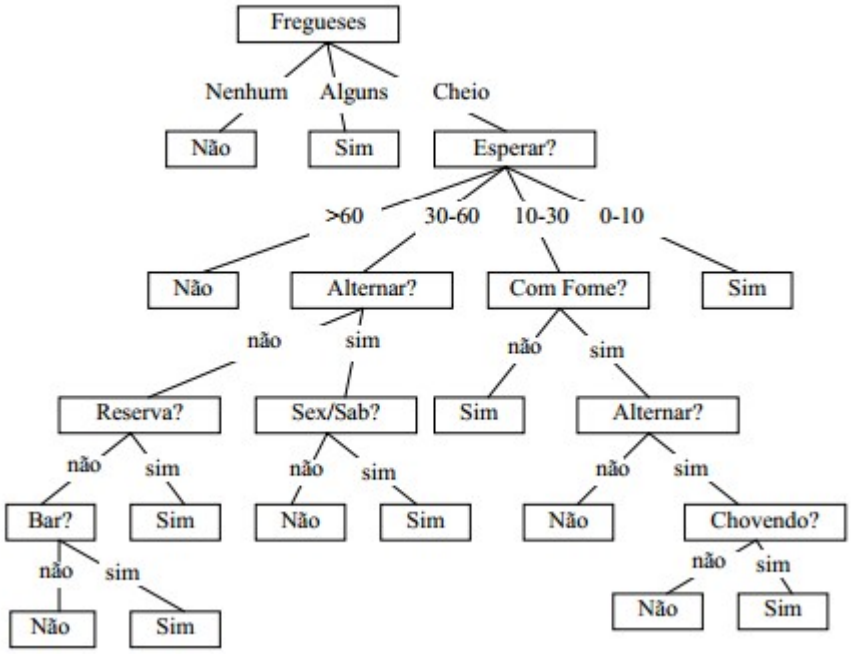

Figura 3.1 Árvore de Decisão para problema de espera para um jantar em um restaurante (POZZER, 2006)

Analisando a figura acima, algumas regras de classificação que podemos perceber são: se não há fregueses, então o cliente não deverá esperar; se o restaurante estiver cheio, a estimativa de espera for entre 30-60 minutos, então se deve analisar a possibilidade de alternar o restaurante, se o cliente não escolher alternar o restaurante e se dia for sexta ou sábado, o cliente deverá esperar para jantar.

Existem diversos algoritmos que realizam a construção de Árvores de Decisão, entre eles C4.5 e ID3.

O C4.5 utiliza a mesma estrutura de árvore descrita anteriormente. O critério de divisão utilizado por este, baseia-se no ganho de informação. O atributo com o maior ganho de informação é escolhido para ser o nó, e, em seguida, as subárvores são construídas com os atributos restantes.

Para criar uma árvore de decisão confiável é necessário verificar se todas as amostras do conjunto de dados pertencem a uma única classe. Caso isso aconteça, a árvore possuirá apenas um nó, e todos os dados que se deseja classificar serão classificados em uma única classe.

3.1.1 ID3

O algoritmo ID3 utiliza o conceito de entropia citado anteriormente para encontrar o atributo que reduz em maior valor a entropia do conjunto de dados, que é o atributo que possui maior ganho de informação. Depois de ter sido encontrado, esse atributo é o novo nó raiz da árvore de decisão e as subárvores desta são obtidas de forma recursiva por esse algoritmo. Esse algoritmo é muito simples, o que facilita o entendimento do processo de construção da árvore, porém, não possui nenhuma forma para tratar valores desconhecidos e o conjunto de dados de entrada deve possuir atributos discretos.

Há diversas diferenças entre o C4.5 em relação ao ID3, as quais são descritas a seguir: manipulação de atributos contínuos, amostras sem alguns valores para determinados atributos, atributos com diferentes custos, e poda de árvore após a sua criação. Apesar destas melhorias, o framework FAMA não possui o algoritmo classificador C4.5, com isso, será utilizado o algoritmo de classificação ID3 para realização dos testes.

3.1.2 Random Forest

Random Forest é um classificador que consiste em uma coleção de classificadores estruturados em árvores da forma $\{h(\boldsymbol{x}, \theta_k), k = 1, ...\}$, em que $\{\theta_k\}$ são vetores independentes identicamente distribuídos e cada árvore vota com uma unidade na classe mais popular para uma entrada **x**. (BREIMAN, 2001)

O Random Forest constrói diversas Árvores de Decisão, e a partir destas um dado é classificado por todas as árvores construídas. Cada árvore determina uma classificação para este dado, o que representa um voto da árvore para essa classe. Após todas as árvores realizarem o processo descrito, o algoritmo Random Forest escolhe a classificação mais popular, ou seja, a classificação que obteve mais votos.

A construção das árvores segue os seguintes critérios: conjuntos de treinamento são criados por amostragem aleatória N’ ≤ N vezes com substituição, em que N é o número de casos no conjunto de treinamento; se existem M variáveis de entrada, um número m << M é especificado para cada nó, m variáveis são selecionadas de forma aleatória e a que melhor divide o conjunto de m variáveis é usado para separar o nó; não há poda. (BREIMAN, 2001)

Algumas vantagens desse algoritmo são: suporta largo banco de dados; estima quais variáveis são importantes na classificação; possui um método eficiente para estimar dados que estão faltando e mantém a acurácia quando uma grande quantidade de dados estão faltando; informa relações entre variáveis e classificação; possui a capacidade de ser estendido para dados não rotulados; entre outras vantagens. (BREIMAN, 2001)

3.2 SVM

As Máquinas de Vetores de Suporte (SVM, Support Vector Machines) são um conjunto de métodos de aprendizado supervisionado usados para classificação e análise de regressão.

As SVMs são embasadas pela teoria de aprendizado estatístico. Essa teoria estabelece condições matemáticas que auxiliam na escolha de um classificador particular a partir de um conjundo de dados de treinamento, de forma que o classificador obtido seja capaz de prever corretamente a classe de novos dados do mesmo domínio em que o aprendizado ocorreu. (LORENA & DE CARVALHO, 2007)

Resumidamente, o algoritmo de uma SVM consiste em dividir o conjunto de dados de entrada geometricamente de acordo com os valores de uma função de decisão. Essa função de decisão basicamente separa valores positivos de negativos.

O processo de treinamento desse algoritmo consiste em treinar um classificador de forma que este aprenda a mapear $x \rightarrow y$ utilizado alguns exemplos de treinamento $\{x_i, y_i\}$, fazendo com que este mapeamento siga a mesma distribuição de probabilidade dos exemplos de treinamento, de acordo com (OGURI, 2006), em que os dados de treinamento $\{x_i, y_i\}$, i = 1,2,3,..., D, para D exemplos de treinamento, x_i é um representação vetorial de um documento e $y_i \in \{-1,1\}$, possuem uma distribuição de probabilidade desconhecida Pr(x,y).

As SVMs serão ditas lineares se sua função de decisão tiver um comportamento linear ou serão ditas não-lineares, caso esta função seja não-linear. As SVMs lineares são eficazes na classificação de conjuntos de dados linearmente separáveis ou que possuam uma distribuição aproximadamente linear. Entretanto, nem sempre é possível dividir os dados de treinameto de forma que formem um hiperplano, para tratar estes casos Bernhard E. Boser, Isabelle M. nGuyon and Vladimir N. Vapnik sugeriram um modo de criar classificadores não-lineares.

3.2.1 SVMs não-lineares

As SVMs lidam com problemas não-lineares mapeando o conjunto de treinamento de seu espaço original, referenciando como entradas, para um novo espaço de maior dimensão, denominado espaço de características.

A única informação necessária sobre o mapeamento é de como realizar o cálculo de produtos escalares entre dados nesse espaço de características, isso é obtido com o uso de funções denominadas Kernels, dessa forma um kernel K é uma função que recebe dois pontos x_i e x_j do espaço de entradas e computa o produto escalar desses dados. (LORENA & DE CARVALHO, 2007)

Tem-se:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \varphi(\mathbf{x_i}) \cdot \varphi(\mathbf{x_j})$$

A utilidade dos Kernels está na simplicidade de seu cálculo e em sua capacidade de representar espaços abstratos. Os Kernels mais utilizados na prática são os polinomiais, os gaussianos e os sigmoidais, listados abaixo:

a) Polinomial

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

b) Gaussiano

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2), \text{ para } \gamma > 0.$$

c) Sigmoidal

$$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c)$$, para algum $\kappa > 0$ and $c < 0$

3.3 Validação Cruzada

A validação cruzada é uma técnica que consiste em particionar um conjunto de dados em subconjuntos mutuamente exclusivos, com o intuito de, usando alguns desses subconjuntos, estimar alguns parâmetros do problema a ser solucionado, e de, utilizando os subconjuntos restantes, testar a solução obtida pela rotina selecionada e avaliar esta solução.

Existem diversos métodos de validação cruzada: método *holdout*, *k-fold*, e *leave-one-out*. Com a finalidade de realizar a validação cruzada na rotina SVM do *framework* FAMA, será dado enfoque no método de validação do *k-fold*.

O método *k-fold* consiste em dividir o conjunto de dados em *k* subconjuntos mutuamente exclusivos de mesmo tamanho. Apenas um subconjunto é utilizado para treinamento, enquanto os *k-1* subconjuntos restantes são utilizados para validação do modelo.

O processo descrito anteriormente é realizado *k* vezes, variando sempre a forma de escolha destes subconjuntos para as tarefas de teste e predição, sempre estabelecendo que cada um dos *k* subconjuntos sejam utilizados exatamente uma vez como dado de teste para a validação do modelo. Após isso, calcula-se a acurácia sobre os erros encontrados. (MELONI, 2009)

3.4 FAMA

O *framework* de Aprendizado de Máquina (FAMA) é um *framework* que tem como objetivo classificar textos por meio do aprendizado de máquina. Nesse *framework*, os dados de entrada devem ser previamente rotulados.

A linguagem utilizada para o desenvolvimento deste *framework* foi C++. Foram contruídas quatro classes com funções específicas são elas: avaliador, treinador, classificador e corpus.

A classe corpus é responsável por armazenar o texto que deverá ser classificado. Cada palavra do texto é armazenada juntamente com seus diferentes tipos de classificação em uma matriz, já que o programa utiliza mais de um algoritmo de classificação, formando assim uma tupla.

Depois de carregar o texto a partir da classe corpus, esses dados são transferidos para a classe treinador, a qual executará os algoritmos de aprendizado, formando assim as regras de classificação. Os resultados obtidos por essa classe são enviados para a classe classificador, a qual é responsável por classificar qualquer outro texto.

A classe avaliador tem como principal função avaliar a classificação final obtida do conjunto de dados fornecido pela classe corpus. Isso é feito pela função *calcularDesempenho* existente nessa classe.

Uma nova classe foi adicionada ao FAMA, a classe abstrata Validador, cuja função é a execução do experimento de aprendizado de máquina. Existem 4 possíveis instanciações dessa classe: ValidadorTreino, ValidadorTeste, ValidadorDivisão, ValidadorKDobras. Estas possuem um método virtual *executarExperimento* o qual é o método que realizará a validação diferente para cada instanciação cujos parâmetros são objetos da classe Treinador e Corpus. Além disso, todas recebem uma referência à classe Avaliador, já que há necessidade de um critério de avaliação.

O ValidadorTreino realiza o treino e classifica a mesma Corpus, avaliando o resultado por meio do Avaliador que tem referência.

O ValidadorTeste é idêntico ao ValidadorTreino, exceto que aplica o conhecimento de classificação a um Corpus distinto, inicializado no construtor.

O ValidadorDivisão realiza uma divisão do Corpus por meio de chamada ao novo método splitCorpus da classe Corpus que tem como parâmetro a porcentagem de divisão. A partir disso é realizado o treino com apenas uma porcentagem da Corpus, e o restante da Corpus é usada para realizar a classificação, sendo realizada a avaliação em seguida. O parâmetro numeroIterações determina a quantidade de vezes que este processo deverá se repetir.

O ValidadorKDobras é a instância responsável por realizar a validação cruzada.

A figura abaixo é o diagrama UML da atual estrutura do *framework*, apresentando os métodos específicos de cada classe descrita anteriormente juntamente com os seus respectivos atributos.

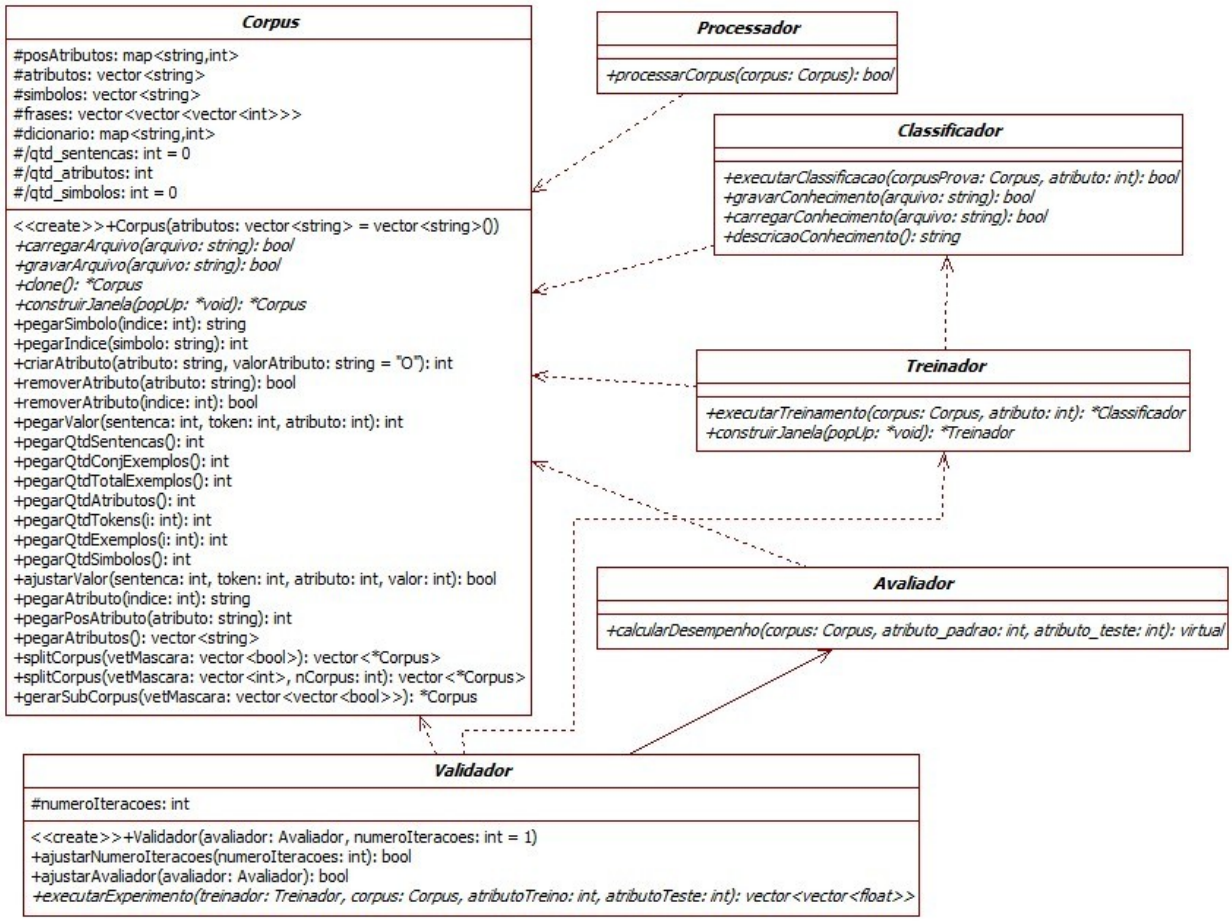

Figura. 3.2 Diagrama de classes do FAMA

Dentre os algoritmos de aprendizado de máquina já implementados no *framework* FAMA estão o MaisProvável, o HMM, o SVM, o TBL, o Naive Bayes, o ID3e o Random Forest.

4 CLASSIFICAÇÃO AUTOMÁTICA DE MALWARE

A criação de uma classificação automática de malware foi dividida em três etapas: conformação dos dados, processamento e validação.

4.1 Conformação dos dados

Como foi citado anteriormente, neste trabalho são utilizados resultados das análises estáticas e dinâmicas de códigos maliciosos, para realizar experimentos de classificação de *malware* utilizando algoritmos de aprendizado de máquina. A etapa de agrupamento dos dados das análises estática e dinâmica foi denominada conformação dos dados, pois consiste em uma etapa em que o conjunto de dados foi tratado por um algoritmo externo ao *framework* FAMA para que fosse possível a utilização deste conjunto nos algoritmos de aprendizado de máquina presentes no *framework*.

As informações da análise estática são obtidas utilizando um programa implementado pela aluna Kizzy em um trabalho de pesquisa anterior, que tratava sobre análise estática de executáveis maliciosos. O software recebe como entrada um diretório com os arquivos binários a serem analisados e para cada código malicioso, é criado um arquivo texto com os seguintes campos: *TimeDateStamp*, *CheckSum*, *NumberOfSections*, *NumberOfRVAandSizes*, *Hash SHA1*, *Hash MD5*, *AddressOfEntryPoint*, *EntryPoint(section)*, *FileDescription*, *Original Name*, *Product Name*, *Company Name*, *Build*, *Version number*, *Import Directory*, *NumberOfSuspectAPI*, *Sections*, *NumberOfSuspectSections*, *Pasta*. Esses campos foram escolhidos, pois representam os principais atributos utilizados para classificar binários em códigos maliciosos, conforme seção 2.1.1.7 A presença dos campos *Hash SHA1* e *Hash MD5* serve para identificar o binário que está sendo analisado. A tabela 4.1 abaixo apresenta as informações contidas no arquivo texto criado para o executável *7zG.exe*. A coluna nomeada como “SUSPEITO” relaciona os valores obtidos para cada atributo com os valores estabelecidos na tabela 2.4, com isso o campo “TRUE” significa que o código binário apresenta um comportamento suspeito para este atributo, e o campo “FALSE” representa um comportamento normal para respectivo atributo.

Foram analisados 6104 códigos, dentre os quais 1461 eram benignos e 4643 eram malignos.

ATRIBUTO	Valor	Valor Estimado	SUSPEITO
TimeDateStamp	04/02/2009	Não se aplica	FALSE
CheckSum	00000000	0003C60F	TRUE
NumberOfSections	0004	Não se aplica	FALSE
NumberOfRVAandSizes	00000010	Não se aplica	FALSE
Hash SHA1	A583EF39D045AD1CDAADA9730EF4455CD09F04D1		
Hash MD5	1807EABB2016AC19BCF022A5597B313F		
AddressOfEntryPoint	00022496	Não se aplica	Não se aplica
EntryPoint(section)	.text	Não se aplica	FALSE
Pasta	3	Não se aplica	Não se aplica
Version Info			
File Description	7-zip GUI	Não se aplica	Não se aplica
Original Name	7zg.exe	Não se aplica	Não se aplica
Product Name	7-zip	Não se aplica	Não se aplica
Company Name	Igor Pavlov	Não se aplica	Não se aplica
Build	0	Não se aplica	Não se aplica
Version number	4.65	Não se aplica	Não se aplica
Import Directory			
NumberOfSuspectAPI	9	Não se aplica	TRUE
Sections			
NumberOfSuspectSections	0	Não se aplica	FALSE

Tabela 4.1 Análise estática do *7zG.exe*

Para agrupar os dados da análise estática gerados para cada um dos executáveis, foi implementado um algoritmo que realiza a leitura automatizada desses dados agrupando-os em uma única tabela. A tabela gerada contém os atributos citados anteriormente juntamente com os seguintes campos: *TimeStamp Verificada*, *CheckSum Calculado*, *CheckSum Verificado*, *NumberOfSections Verificado*, *NumberOfRVAandSizes Verificado*. Esses novos atributos foram criados para indicar se os atributos correspondentes estão de acordo com as restrições especificadas na seção 2.1.1.7, ou seja, o atributo *TimeDateStamp Verificada* indicará se o atributo *TimeDateStamp* é suspeito, com valor VERDADEIRO caso seja e FALSO, caso contrário. A tabela 4.2 a seguir é um trecho da tabela que reúne os dados da análise estática de *malware* descrita anteriormente.

TimeDateStamp	TimeDateStamp Verificada	CheckSum	CheckSum Calculado	CheckSum Verificado	NumberOf Sections	NumberOf Sections Verificado
02/04/2009	FALSO	00000000	0006A4C3	VERDADEIRO	4	FALSO
07/11/2004	FALSO	0000CF75	0000440E	VERDADEIRO	3	FALSO

Tabela 4.2 Tabela de dados

Posteriormente, uma segunda tabela foi criada com os dados obtidos na análise dinâmica do mesmo conjunto de binários citado anteriormente (DE ANDRADE, 2013). A partir da análise dinâmica, para cada um dos códigos maliciosos, um arquivo do tipo *.csv* foi criado. Nesse arquivo, cada linha armazenava o nome de uma API chamada pelo binário. O algoritmo implementado para gerar essa segunda tabela, realiza a leitura automatizada dos arquivos *.csv* mencionados a fim de calcular quantas vezes cada API foi chamada. Nessa tabela, foram colocadas como atributo-coluna todas as APIs que foram chamadas pelo menos uma vez por qualquer um dos binários, e cada linha, por sua vez, corresponde a um determinado binário. Além disso, as linhas serão preenchidas com o número de vezes que a API correspondente a uma determinada coluna foi chamada pelo binário correspondente a uma determinada linha. A tabela 4.3 é um trecho da tabela descrita que contém os dados da análise dinâmica. A figura 4.1 ilustra as informações contidas no arquivo *.csv* gerado pelo Cuckoo sandbox na análise dinâmica do binário *7zG.exe*.

	"CreateMutexW"	"CreateFileW"	"ReadFile"	"WriteFile"	"RegOpenKeyW"	"CreateRemoteThread"
winUpdate32Login.exe	6	1	0	0	0	0
SYSTEM.DLL	7	15	115	1	125	4
Bacterio61.exe	7	1	1	1	11	0

Tabela 4.3 Tabela de pré-processamento – Análise dinâmica

20120621112410134,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
CTF.LBES.MutexDefaultS-1-5-21-2052111302-1993962763-1957994488-1003","hMutex->0x00000078"

20120621112410144,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
CTF.Compart.MutexDefaultS-1-5-21-2052111302-1993962763-1957994488-1003","hMutex->0x00000084"

20120621112410254,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
CTF.Asm.MutexDefaultS-1-5-21-2052111302-1993962763-1957994488-1003","hMutex->0x0000008c"

20120621112410274,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
CTF.Layouts.MutexDefaultS-1-5-21-2052111302-1993962763-1957994488-1003","hMutex->0x00000094"

20120621112410404,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
CTF.TMD.MutexDefaultS-1-5-21-2052111302-1993962763-1957994488-1003","hMutex->0x0000009c"

20120621112410484,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
CTF.TimListCache.FMPDDefaultS-1-5-21-2052111302-1993962763-1957994488-1003MUTEX.DefaultS-1-5-21-
2052111302-1993962763-1957994488-1003","hMutex->0x000000a4"

20120621112414690,"196","3f317b522f0bc19ac1620bbea0718.exe","CreateMutexW","lpName->
MSCTF.Shared.MUTEX.EIF","hMutex->0x000000b4"

Figura 4.1 Relatório gerado pelo Cuckoo SandBox para o arquivo 7zG.exe

Em seguida, foi realizada a união das tabelas que continham os dados da análise estática e dinâmica em uma terceira tabela, em que cada linha representava um binário e cada coluna representava um atributo da análise estática ou da análise dinâmica, citado anteriormente.

4.2 Processamento

Após a etapa de conformação de dados, se tornou necessário realizar um processamento dos dados obtidos na contagem das APIs para cada binário, pois era necessário diminuir a quantidade valores obtidos para a contagem de APIs, já que um dos algoritmos de classificação utilizados do framework será o ID3 que não suporta conjunto de dados contínuos e há atributos com campos não numéricos, o que não é aceito pelos algoritmos de classificação pertencentes ao framework. Esse processamento possibilita uma análise quantitativa destes dados e também, facilita o tratamento destes dados pelo classificador de *malware*. Vale evidenciar que essa etapa de processamento dos dados é realizada dentro da instanciação do *framework* FAMA.

O objetivo do algoritmo de processamento implementado é discretizar os valores contínuos da tabela final gerada pelo conformação dos dados, além de diminuir os valores

possíveis e permitir a utilização dos algoritmos de aprendizado de máquina na instanciação do framework FAMA, para isso aplica-se uma função logaritmo nesses valores.

A rotina implementada para processar os dados recebe como entrada uma lista de atributos e um objeto da classe *Corpus* que continha a tabela de dados das análises estática e dinâmica.

Com as informações da entrada, aplica-se a função logaritmo de base 2 nos dados de interesse e os resultados obtidos para cada coluna da tabela de dados são armazenados em novas colunas ao final dessa tabela.

Além desse processamento, foi implementado um segundo processamento para transformar atributos discretos que possuem como domínio os valores “VERDADEIRO” e “FALSO” em atributos cujo domínio é 0 e 1.

A entrada dessa rotina eram uma lista dos atributos que serão transformados e o corpus utilizado no FAMA que continha a tabela de dados das análises estática e dinâmica. A partir dessas informações, para cada atributo pertencente ao primeiro arquivo de entrada, é criado um novo atributo cujo domínio é 0 e 1, sendo que se o primeiro atributo tiver como valor “VERDADEIRO”, o novo atributo terá o valor 1, o mesmo vale para “FALSO” e 0. Esses novos atributos são adicionados ao final do conjunto de dados.

4.3 Validação

Foram realizados três experimentos com instanciações do FAMA utilizando os algoritmos de aprendizagem de máquina SVM, Random Forest e ID3. Esses algoritmos foram escolhidos para a realização dos experimentos de classificação de *malware* por terem sido previamente implementados para o *framework* FAMA por outros alunos. Para que fossem utilizados outros algoritmos de aprendizado de máquina iria ser necessário implementá-los, entretanto esta tarefa não está no escopo desta pesquisa.

Para a realização dos experimentos foram utilizados 6104 códigos, sendo 1461 códigos benignos e 4643 códigos maliciosos.

Após a realização dos processamentos citados anteriormente, foram selecionados os atributos que iriam ser utilizados para o aprendizado do SVM e do ID3, os quais foram: *Maligno*, *logCreateMutexW*, *logOpenMutexW*, *logCreateFileW*, *logReadFile*, *logWriteFile*, *logRegOpenKeyW*, *logCreateRemoteThread*, *logRegSetValueExW*, *logRegEnumKeyExW*,

logFindWindowW, *logCreateProcessA*, *logDeleteFileW*, *logWriteProcessMemory*, *logTerminateProcess*, *logControlService*, *logOpenSCManagerW*, *logOpenServiceW*, *logRegEnumValueW*, *logRegCreateKeyW*, *logCreateProcessW*, *logReadProcessMemory*, *logShellExecuteExW*, *logRegDeleteKeyW*, *logURLDownloadToFileW*, *logCreateServiceA*, *logDeleteService*, *logStartServiceW*, *logCreateServiceW*, *NewTimeDateStampVerificado*, *NewCheckSumVerificado*,*NewNumberOfSectionsVerificado*,N*ewNumberOfRVAandSizesVerificado*. Em que os atributos que possuem o prefixo *log* correspondem a atributos que foram discretizados e os que possuem o prefixo *New* correspondem a atributos transformados, segundo foi explicado anteriormente, e possuem como domínio 0 e 1. Os atributos correspondentes à análise dinâmica representam APIs comumente utilizadas por *malware* as quais foram catalogadas em (DE ANDRADE, 2013). As APIs relacionadas à análise estática foram escolhidas de acordo com a seção 2.1.1.7. Com isso, o conjunto de dados adquiriu o formato desejado para a aplicação dos algoritmos SVM, Random Forest e ID3.

Em seguida, foram realizados os experimentos com os algoritmos de aprendizado de máquina selecionados. Estes algoritmos criavam regras de classificação a partir do conjunto de dados obtido anteriormente e, consecutivamente, reclassificava cada código malicioso do conjunto de dados. Essa classificação era armazenada em um novo atributo adicionado ao final do conjunto de dados.

A fim de comparar os algoritmos utilizados, foi calculada a acurácia de ambos os experimentos, que consiste em comparar a classificação produzida pelos classificadores dos algoritmos selecionados com os rótulos de classificação do conjunto de dados. A fórmula da acurácia é descrita por:

$$(total\ de\ acertos)/(quantidade\ de\ dados) \quad \textbf{(4.3.1)}$$

Além disso, o tempo total de execução é o tempo que as rotinas do SVM e ID3 demoram a carregar o conjunto de dados inteiro, aplicar o algoritmo de classificação, avaliar o resultado, gravar o novo conjunto de dados em memória e gravar a descrição do conhecimento obtido.

Para cada um dos experimentos apresentado nas seções que seguem foi utilizada uma instanciação diferente do *framework* FAMA. Sua principal diferença está nos classificadores, os quais utilizam algoritmos de aprendizado de máquina distintos.

4.3.1 Experimento 01

Esse experimento foi realizado com o algoritmo de aprendizado de máquina SVM. Nessa instanciação do FAMA foi utilizada a biblioteca LIBSVM desenvolvida para máquinas de vetores de suporte, cujos parâmetros foram fixados com os seguintes valores abaixo:

cost	nu	epsilon	svm_type	degree	gamma (1/k)	coef0	Cache size (MB)
1	0,5	0,1	C-SVC	3	0,5	0	256

Tabela 4.4 Tabela dos parâmetros do SVM

O experimento consistiu em realizar a classificação para três conjuntos de dados: o conjunto de dados de ambas as análises estática e dinâmica, o conjunto de dados apenas com a análise dinâmica e o conjunto de dados apenas com a análise estática. Para cada conjunto de dados foi calculada a acurácia, pela validação cruzada com índice k= 10, para a classificação obtida para os quatro tipos de kernel estudados, assim como foi calculado o tempo de execução do algoritmo. Os resultados estão representados na ttabela 4.5.

	Todos		Dinâmica		Estática	
	Tempo(s)	Acurácia(%)	Tempo(s)	Acurácia(%)	Tempo(s)	Acurácia(%)
Linear	58,9	76,06	55,0	76,06	11,9	76,06
Polinomial	11000,5	77,05	10820,2	76,38	82,5	76,06
Gaussiano	78,7	78,98	57,3	78,54	20,7	76,06
Sigmoid	44,1	75,15	28,3	72,79	29,5	75,90

Tabela 4.5 Tabela de resultados do teste com SVM

Analisando a tabela 4.5 acima é possível perceber que para a maioria dos kernel utilizados, a utilização de dados da análise dinâmica apresenta melhores resultados que a utilização de dados da análise estática. Além disso, é possível notar que mesmo que o kernel polinomial apresente uma acurácia próxima à acurácia obtida pelo kernel gaussiano, o tempo de execução do primeiro é significativamente maior que este último. Em adição às observações anteriores, é notável que o kernel que apresenta melhores acurácias por tempo é o kernel gaussiano.

4.3.2 Experimento 02

O segundo experimento utiliza o algoritmo de aprendizado de máquina Random Forest. Para utilizar este algoritmo é necessário definir a quantidade árvores que deverão ser criadas e a quantidade de atributos que serão utilizados para construir essas árvores.

A primeira coluna representa a quantidade de árvores que foram construídas. Assim como no primeiro experimento, foram realizados experimentos com conjunto de dados de ambas as análises estática e dinâmica, conjunto de dados apenas com a análise dinâmica e conjunto de dados apenas com a análise estática. Foi estabelecido que a quantidade de atributos utilizados para construir as árvores seria 70% dos atributos totais, o que consistem em 24, 21 e 3, para os respectivos conjuntos citados anteriormente. Os dados obtidos deste experimento seguem na tabela 4.6.

N	Todos		Dinâmica		Estática	
	Tempo(s)	Acurácia(%)	Tempo(s)	Acurácia(%)	Tempo(s)	Acurácia(%)
3	78,301s	82,08%	60,484s	81,88%	3,330s	76,06%
5	136,275s	82,04%	106,263s	81,86%	5,478s	76,08%
7	176,816s	82,11%	150,511s	81,90%	7,401s	76,08%
9	228,276s	82,42%	180,900s	81,78%	9,618s	76,08%
11	276,973s	82,18%	231,615s	81,80%	11,572s	76,06%

Tabela 4.6 Tabela de resultados do Random Forest

Em comparação ao experimento anterior, é perceptível que este experimento apresenta melhores acurácias. Além disso, a utilização de dados da análise dinâmica possui melhor acurácia em relação a utilização de dados da análise estática.

4.3.3 Experimento 03

O terceiro experimento utiliza o algoritmo de aprendizado de máquina ID3.

Assim como nos outros experimentos, foram realizados experimentos com conjunto de dados de ambas as análises estática e dinâmica, conjunto de dados apenas com a análise

dinâmica e conjunto de dados apenas com a análise estática. Os resultados obtidos deste experimento seguem na tabela 4.6.

	Todos		Dinâmica		Estática	
	Tempo(s)	Acurácia(%)	Tempo(s)	Acurácia(%)	Tempo(s)	Acurácia(%)
ID3	36.218s	82,67%	30,050s	82,32%	1.304s	76,08%

Tabela 4.7 Tabela de resultados do ID3

Em comparação com os experimentos anteriores, este experimento apresentou melhores acurácias para menores tempos de execução. Além disso, a utilização de dados da análise dinâmica possui melhor acurácia em relação a utilização de dados da análise estática.

4.3.4 Análise Comparativa dos Experimentos

A tabela 4.8 apresenta os melhores resultados para cada um dos experimentos realizados.

	Acurácia (%)	Acurácia(%)
ID3	82,67	-
SVM	78,98	-
Random Forest	82,42	91,1*

Tabela 4.8 Tabela de melhores resultados de cada experimento

Analisando a tabela 4.8 juntamente com as tabelas 4.5, 4.6 e 4.7, é possível perceber que todas as melhores acurácias listadas pertencem à Corpus que utilizava dados da análise estática e análise dinâmica. A melhor acurácia do SVM corresponde ao SVM com kernel do tipo gaussiano, e o algoritmo de aprendizado de máquina que apresentou melhor acurácia foi o ID3.

Além disso, em todos os resultados listados, é notável que a utilização de dados da análise estática e dinâmica apresenta melhor desempenho em relação aos resultados com dados apenas da análise dinâmica, o que significa que analisar atributos estáticos de um binário melhorar significativamente a classificação destes.

Comparando os resultados obtidos com os experimentos aqui citados com o experimento em (DE ANDRADE,2012), que esta representado na tabela 4.8 com um asterisco, é possível observar que houve uma diminuição de acurácia em torno de 10%, o que pode ser uma

consequência da utilização de uma Corpus diferente. Contudo, como ambos experimentos utilizaram o algoritmo de aprendizado de máquina Random Forest presente no FAMA, espera-se que o aumento de aproximadamente 0,25% apresentado na tabela 4.7 ao adicionar dados da análise estática aos da dinâmica ocorra no experimento em (DE ANDRADE, 2012) caso a este sejam adicionados os dados da análise estática.

Em adição aos resultados anteriores, é importante salientar a grande quantidade de tempo utilizada na execução do SVM com kernel polinomial, o qual não produziu as melhores acurácias.

5 CONCLUSÃO

O aumento diário do número de ameças virtuais evidencia a necessidade de pesquisas sobre técnicas eficientes de identificação de *malwares.* Nesse contexto, insere-se também a importância dos algoritmos de aprendizado de máquina, pois aplicados a classificação de *softwares* maliciosos, permitem que esse processo de classificação seja otimizado, sem que haja a necessidade de um ser humano com conhecimentos especializados nessa área.

Assim, a construção de um identificador de executáveis malignos que utilize algoritmos de aprendizado de máguina, permite que a própria máquina verifique os relatórios obtidos a partir de análises estáticas e dinâmicas desses executáveis e os classifique.

Esse trabalho insere-se nessa área de pesquisa, tendo como principal objeto de estudo algoritmos de aprendizado de máquina que podem ser utilizados na classificação de *malware*. Além disso, nessa pesquisa foram realizadas instanciações do framework FAMA com dois algoritmos de aprendizado de máquina, ID3, Random Forest e SVM, com intuito de verificar o desempenho desses classificadores.

A principal contribuição desta pesquisa está no uso do conjunto de dados obtidos na análise dinâmica associados ao dados obtidos na análise estática. Em todos os experimentos realizados, a utilização desses conjuntos de dados agrupados resultou em melhores acurácias, evidenciando que os dados gerados estaticamente têm uma grande importância na identificação dos códigos maliciosos, ainda que sejam obtidos quando o *malware* não está em execução.

Além disso, pode-se destacar que os experimentos realizados mostram que o algoritmo de aprendizado de máquina ID3 apresentou melhores acurácias. Pode-se notar também, que nos experimentos realizados utilizando-se os dados da análise dinâmica foram obtidas melhores taxas de acurácia quando comparadas às obtidas utilizando-se aos da análise estática.

Para trabalhos futuros, recomenda-se que sejam implementados outros algoritmos de aprendizado de máquina de forma a possibilitar a análise do desempenho de outros classificadores. Pode ser interessante, também, aprimorar a seleção dos atributos que serão utilizados no aprendizado do SVM e do ID3, bem como aumentar a amostra de códigos maliciosos e não-maliciosos submetidos à análise dinâmica e à análise estática para que essa represente de forma cada vez mais fidedigna o universo de *malwares* existente.

6 REFERÊNCIAS BIBLIOGRÁFICAS

BREIMAN, LEO. **Random Forests.** Paper – Universidade da Califórnia, Berkeley, 2001.

CONDUTA, BRUNO C. **Aprendizado de Máquina**. Dissertação (Mestrado em Inteligência Artificial) - Faculdade de Tecnologia da Universidade Estadual de Campinas, 2010.

DA SILVA, LUIZA. **Uma Aplicação de Árvores de Decisão, Redes Neurais e KNN para a Identificação de Modelos ARMA Não-Sazonais e Sazonais**. Tese (Pós-graduação em Engenharia Elétrica) - PUC-Rio, 2005.

DE ANDRADE, CESAR A. B. **Análise Automática de Malwares Utilizando as Técnicas de Sandbox e Aprendizado de Máquina**. Dissertação (Mestrado em Sistemas e Computação) - Instituto Militar de Engenharia, 2013.

DE SÁ, NELSON. **General detalha implantação do Centro de Defesa Cibernética, novo órgão brasileiro.** Folha de S. Paulo [online], São Paulo, 7 maio 2012. Homem. Disponível: http://www1.folha.uol.com.br/tec/1085498-general-detalha-implantacao-do-centro-de-defesa-cibernetica-novo-orgao-brasileiro.shtml [capturado em 20 out. 2012].

DUMMER, DANIEL. **Windows Portable Executable**. Devmedia [online], São Paulo, set. 2006. Disponível: http://www.devmedia.com.br/articles/viewcomp.asp?comp=857 [capturado em 5 set. 2012].

GAMA, JOÃO M. P. **Árvores de Decisão Adaptativas.** Dissertação (Mestrado em Inteligência Artificial e Computação) - Universidade do Porto, 2004.

GUILHERME, PAULO. **O que é Sandbox?**. Tecmundo [online], São Paulo, 5 jul. 2012. Disponível: http://www.tecmundo.com.br/spyware/1172-o-que-e-sandbox-.htm [capturado em 19 ago. 2012].

KLEINA, NILTON. **Centro de Defesa Cibernética, a nova arma do Brasil contra invasões hacker.** Tecmundo [online], São Paulo, 7 maio 2012. Disponível: http://www.tecmundo.com.br/seguranca-de-dados/23193-centro-de-defesa-cibernetica-a-nova-arma-do-brasil-contra-invasoes-hacker.htm [capturado 20 out. 2012].

LIGH, MICHAEL H.; ADAIR, STEVEN; HARTSTEIN, BLAKE; RICHARD, MATTHEW. **Malware Analysts Cookbook and DVD: Tools and techniques for fighting malicious code.** Wiley Publishing, Inc. Indianapolis, Estados Unidos. 2011

LORENA, ANA CAROLINA; DE CARVALHO, ANDRÈ C. P. L. F. **Uma Introdução às *Support Vector Machines*.** Paper publicado em Revista de Informática Teórica e Aplicada (RITA), Volume XIV, Número 2, 2007.

MELONI, RAPHAEL. **Classificação de Imagens de Sensoriamento Remoto usando SVM**. Dissertação (Mestrado em Informática) – PUC-Rio, 2009.

MITCHELL, T. M. **Machine Learning**. McGraw-Hill, Inc., New York, Estados Unidos, 1 ed. 1997.

OLIVEIRA, FELIPE E DO NASCIMENTO, JOILSON. **Algoritmos de aprendizado de máquina para tarefas de classificação morfossintática.** Iniciação à Pesquisa, Instituto Militar de Engenharia, Rio de Janeiro, 2012.

POZZER, C.T. **Aprendizado por Árvores de Decisão.** Notas de aula, Universidade Federal de Santa Maria, Rio Grande do Sul, 2006.

QUINLAN, J. R. Rulequest [online], fev. 2012. Disponível: http://rulequest.com/see5-comparison.html [capturado 5 set. 2012].

RAMOS, DURVAL. **Cerca de 220.000 vírus são criados por dia**. Tecmundo [online], São Paulo, 16 nov. 2011. Disponível: http://www.tecmundo.com.br/virus/15411-cerca-de-220-000-virus-sao-criados-por-dia.htm [capturado 5 set. 2012].

RUSSEL, S. J.; NORVIG, P. **Artificial Intelligence: A modern approach**. New Jersey: Prentice-Hall, 1995. 1 ed., 932p.

SANTOS, CÍCERO. **Aprendizado de Máquina na Identificação de Sintagmas Nominais: O Caso do Português Brasileiro**. Dissertação (Mestrado em Sistemas e Computação) - Instituto Militar de Engenharia, 2005.

SIMON. H.A. **Why should machines learn?** In Michalski et al.1983.