Integrating Static and Dynamic Malware Analysis Using Machine Learning

R. J. Mangialardo and J. C. Duarte

***Abstract*— Malware Analysis and Classification Systems use static and dynamic techniques, in conjunction with machine learning algorithms, to automate the task of identification and classification of malicious codes. Both techniques have weaknesses that allow the use of analysis evasion techniques, hampering the identification of malwares. In this work, we propose the unification of static and dynamic analysis, as a method of collecting data from malware that decreases the chance of success for such evasion techniques. From the data collected in the analysis phase, we use the C5.0 and Random Forest machine learning algorithms, implemented inside the FAMA framework, to perform the identification and classification of malwares into two classes and multiple categories. In our experiments, we showed that the accuracy of the unified analysis achieved an accuracy of 95.75% for the binary classification problem and an accuracy value of 93.02% for the multiple categorization problem. In all experiments, the unified analysis produced better results than those obtained by static and dynamic analyzes isolated.**

***Keywords*— Information Security, Malware, Static Analysis, Dynamic Analysis, Unified Analysis, Machine Learning.**

I. INTRODUÇÃO

MALWARES (softwares maliciosos) são programas desenvolvidos para executar ações danosas em um computador. Os principais motivos que levam ao desenvolvimento e a propagação de códigos maliciosos são a obtenção de vantagens financeiras, a coleta de informações confidenciais, o desejo de autopromoção e o vandalismo. Além disto, os códigos maliciosos são muitas vezes usados como métodos intermediários e possibilitam a prática de golpes, a realização de ataques e a disseminação de spam [1].

A detecção e a análise de códigos maliciosos são atividades cruciais para qualquer mecanismo de defesa contra estes ataques. Para identificar os *malwares* duas técnicas podem ser empregadas: a análise estática e a análise dinâmica de *malwares*. A análise estática permite extrair características do código sem executá-lo e a análise dinâmica permite extrair informações quando o código é executado [15]. Os desenvolvedores de códigos maliciosos utilizam técnicas antianálise ou de evasão das análises estática e dinâmica para evitar que informações sobre os *malwares* sejam obtidas, dificultando ou impedindo a sua identificação. Além deste problema, ainda existe a demanda de tempo para classificar o código como um *malware* pelo especialista de segurança. Esta tarefa não é simples e para que seja possível são necessários o planejamento e desenvolvimento de sistemas que executem a tarefa de forma automatizada, reduzindo o tempo de resposta de reação à ação dos *malwares*. Técnicas de aprendizado de máquina têm sido utilizadas em conjunto com algoritmos de coleta estática e dinâmica para a identificação e classificação de *malwares*, de tal forma a automatizar a classificação e reduzir o tempo de desenvolvimento de *softwares* de antivírus para combater os códigos maliciosos. Trabalhos de análise de malwares realizados destacam normalmente a aplicação de técnicas isoladas de análise de malwares. A tabela I apresenta alguns trabalhos relacionados de análise de malwares e o método que foi utilizado:

TABELA I. TRABALHOS RELACIONADOS.

AUTOR	ABORDAGEM	PRECISÃO
[16]	DINÂMICA	Random Forest, 93,6%
[23]	DINÂMICA	kNN, 83,90%
[24]	DINÂMICA	Random Forest, 98,3%
[26]	ESTÁTICA	Árvores de Decisão, 98%
[27]	ESTÁTICA	Random Forest, 92,34%
[28]	ESTÁTICA	SVM, 98,73%

Com o objetivo de contribuir com a melhoria do processo de análise e, assim, diminuir o impacto das atividades de evasão, contribuindo com a segurança da informação, este trabalho apresenta um método para a construção de uma base dados unificada para a análise e classificação automática de *malwares*. Outra inovação do trabalho é a classificação em tipos de *malwares*. Classificamos os artefatos experimentando e introduzindo o algoritmo de aprendizado de máquina C5.0 na classificação automática e comparamos seus resultados com os obtidos pelo algoritmo *Random Forest*.

I I. ANÁLISE DE MALWARES

A análise de *malwares* é o processo de identificação e classificação de códigos maliciosos. Com o objetivo de identificar *malwares*, especialistas de segurança adotam um método geral de investigação, que se divide normalmente em 5 fases [3]:

- Fase 1: preservação forense e exame dos dados voláteis;
- Fase 2: exame da memória do sistema infectado;
- Fase 3: análise forense: compreende o exame dos meios de armazenamento permanentes (disco rígido, fitas de *backup*, etc.);
- Fase 4: verificação das características do arquivo desconhecido;
- Fase 5: análise estática e dinâmica do arquivo suspeito.

As fases 4 e 5 são utilizadas para obter as características dos códigos contidos em arquivos. Estes arquivos podem ter vários formatos e, neste trabalho, para manter uma padronização,

R. J. Mangialardo, Instituto Militar de Engenharia (IME), Rio de Janeiro, Rio de Janeiro, Brasil, mangialardo@ime.eb.br

J. C. Duarte, Instituto Militar de Engenharia (IME), Rio de Janeiro, Rio de Janeiro, Brasil, duarte@ime.eb.br

facilitar a observação e tornar a busca por características uniformizada, utilizamos o formato de arquivo Windows PE32. Este formato de arquivo possui especificações abertas, permitindo a investigação de suas características [3]. Estas características são estudadas pelo analista, que então identificará o arquivo como *malware* ou não *malware* e, também, poderá classificá-los em tipos ou famílias de *malwares*.

A análise estática de *malwares* é o processo de análise do código sem executá-lo no sistema e subdivide-se em análise estática básica e análise estática avançada [4]. Na análise estática básica, o analista utiliza programas antivírus, *hashes* para identificar o arquivo, captura de informações existentes em *strings*, funções e cabeçalhos de arquivos. Na análise estática avançada, o código do programa é desmontado, utilizando um desmontador, e é feita uma engenharia reversa com o objetivo de descobrir o que o programa faz.

A análise dinâmica de *malwares* é o processo de análise do código em execução, necessitando de um ambiente isolado, com medidas de segurança adotadas para evitar que outros sistemas sejam contaminados pela execução do código malicioso. Assim como a análise estática, a análise dinâmica divide-se em análise dinâmica básica e análise dinâmica avançada [4]. Na análise dinâmica básica, o *malware* é executado em um ambiente virtualizado e o seu comportamento é monitorado, permitindo obter automaticamente as características de um código malicioso em execução. Também, a análise pode ser feita observando-se os estados do sistema operacional, antes e após a execução do código. Toda a análise baseia-se na observação das chamadas de APIs que são executadas [5].

Entretanto, para evitar o sucesso da análise, os desenvolvedores de *malwares* empregam mecanismos de antianálise. Estes mecanismos se dividem em quatro categorias [6]:

• Ofuscação: técnicas utilizadas para tornar mais difícil a criação de assinaturas e a análise do código desmontado;

• *Antidisassembly*: técnicas utilizadas para comprometer o funcionamento dos desmontadores e/ou prejudicar o processo de desmontagem de código;

• *Anti-VM*: técnicas utilizadas para detectar a presença de máquinas virtuais ou prejudicar seu funcionamento;

• *Antidebugging*: técnicas utilizadas para comprometer o funcionamento de depuradores e/ou o processo de depuração do código dificultando a engenharia reversa.

A classificação de *malwares* pode ser automatizada utilizando uma técnica de inteligência artificial chamada aprendizado de máquina. A combinação das análises estática e dinâmica e a utilização de aprendizado de máquina podem melhorar a capacidade e velocidade de detecção dos *malwares* e é uma possível boa solução para tratar *malwares* blindados, metamórficos, ou aqueles ainda não identificados (*zero-day malwares*).

III. CLASSIFICAÇÃO AUTOMÁTICA DE MALWARES

A classificação automática de malwares, utilizando algoritmos de aprendizado de máquina, envolve duas áreas de conhecimento. A primeira é a da segurança de computadores que trata do problema dos códigos maliciosos, sua natureza, formas de ação, efeitos sobre os sistemas e a busca por soluções que tratem e impeçam suas ameaças [1]. A outra área é a Inteligência Artificial, mais precisamente o aprendizado de máquina, que utilizamos para automatizar o processo de identificação e classificação de *malwares*. *Malwares* podem ser classificados de várias maneiras, existindo várias taxonomias possíveis utilizadas para a classificação.

A busca por uma nomenclatura padronizada de *malware* é tão antiga quanto os vírus de computadores, porém, apesar de algumas iniciativas terem sido tomadas, as empresas que desenvolvem antivírus não possuem obrigatoriedade em seguir qualquer padronização. Um exemplo de sistema criado para padronizar nomes é o esquema de nomeação CARO [4]. Obviamente, a classificação de *malwares* é muito difícil de ser realizada, seja qual for o método de classificação adotado porque os *malwares* não são desenvolvidos com base em padrões e também porque *malwares* modernos possuem grande capacidade de proliferação e mutação [4][6].

Neste trabalho, inicialmente os exemplos são classificados como malignos, para os artefatos identificados como *malwares* e benignos, para aqueles identificados como não *malwares*. Em seguida, utiliza-se a definição de tipos de *malwares* derivada do CERT.BR para classificar os artefatos maliciosos como [1]: vírus, worm, bot, trojan, spyware, backdoor e rootkit. Os tipos de *malwares* são definidos conforme a forma como o *malware* é adquirido, como é instalado, como se propaga e como atua no sistema infectado. Esta classificação adotada não é definitiva, devido à grande dificuldade decorrente da grande proliferação de *malwares*, de sua variabilidade e outras classificações adotadas por organizações e especialistas.

A análise é feita sobre arquivos com o formato Windows PE 32, que é o formato adotado para arquivos executáveis na plataforma Windows. Entretanto, a metodologia empregada pode ser estendida para outros formatos de arquivos empregados por outros sistemas operacionais.

Para realizar a análise estática é necessário utilizar um programa que permite identificar e ler as características dos arquivos e de programas que permitam desmontar o código. Vários softwares e aplicações podem ser utilizados: um exemplo é o programa IDA Pro 5.0, um disassembler cuja licença é proprietária [7]. Programas com código livre também podem ser utilizados, como por exemplo, o programa Pyew, em ambientes Linux [8]. Outro aplicativo que pode ser utilizado, para obter as características de um arquivo com formato Windows PE 32, é o pescanner, desenvolvido em Python [3][9]. Por sua vez, o Exiftool é um aplicativo que permite ler, gravar, e manipular dados em vários formatos.

Para realizar a análise dinâmica pode-se utilizar um aplicativo como o Cuckoo Sandbox [2].

Para realizar a automatização do processo de classificação, identificando os artefatos como *malwares* ou não-*malwares* ou

em tipos de *malwares* ou em famílias de *malwares*, pode-se utilizar o aprendizado de máquina. Um framework de aprendizado de máquina contém uma série de programas de aprendizado de máquina que podem ser escolhidos para a automatização do processo de identificação de *malwares*. São exemplos de algoritmos de aprendizado de máquina, o algoritmo C5.0 e o *Random Forest*.

O C5.0 é um classificador que é representado como uma árvore de decisão ou um conjunto de regras e é derivado dos algoritmos de aprendizado de máquina ID3 [17] e C4.5[18]. O C5.0 divide a amostra tendo como base o atributo que resulta em maior ganho de informação em cada nível da árvore. A árvore, construída pelo C5.0 é uma estrutura de dados recursiva formada por um nó-folha que corresponde a uma classe ou por um nó de decisão que contém um teste sobre algum atributo.

O Random Forest [20] é um classificador baseado em árvores de decisão e pode reconhecer os padrões de várias classes ao mesmo tempo. O algoritmo é do tipo comitê e é composto por diversas árvores, formando assim uma floresta de decisão. Ao final da execução, a classificação de cada um dos exemplos será o resultado da votação dos classificadores e um exemplo receberá a classificação resultante da maioria dos votos dos classificadores.

IV. AMBIENTE DAS ANÁLISES ESTÁTICA E DINÂMICA DE MALWARES

O ambiente de análise criado e configurado para a realização de todos os experimentos é apresentado no diagrama da Fig. 1.

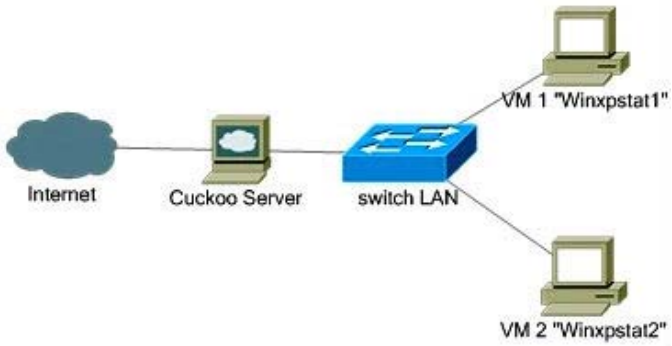

Figura 1. Diagrama do ambiente de análise.

A máquina hospedeira usada (Cuckoo Server) possui 4GB de memória RAM, capacidade para instalar duas máquinas virtuais e 500 GB de espaço de armazenamento secundário. Os softwares utilizados em todas as fases do trabalho são livres, com exceção do sistema operacional Windows XP, instalado na máquina virtual, e que foi utilizado como sistema alvo da execução dos arquivos de malwares e não malwares. Todas as configurações de atualização de software, de firewall e de segurança do Windows XP foram desabilitadas. O Windows XP foi utilizado porque ele é compatível com todos os artefatos analisados, *malwares* e não *malwares*, e também porque com ele foi possível executar facilmente todos os arquivos sem qualquer obstáculo operacional imposto por sistemas ou módulos de segurança implementados nos sistemas operacionais Windows 7 ou 8. Também foi instalado o conjunto de programas de escritório Office 2007 e o programa Acrobat Reader, para o caso de utilização destes softwares por algum *malware*. A máquina hospedeira foi configurada com o sistema operacional Linux, Ubuntu, distribuição 14.04 LTS. O software de virtualização utilizado foi o VirtualBox 4.2.16 Ubuntu r86992. Os softwares e aplicativos utilizados na análise estática foram o exiftool [10], pescanner.py [25] e o programa Pyew [8]. Para a análise dinâmica foi utilizado o Cuckoo Sandbox versão 1.0. Para dificultar o emprego de técnicas anti-vm, configuramos o Cuckoo Sandbox de tal forma a dificultar a utilização de sensores pelo malware. Para realizar esta configuração, baixamos o pacote CuckooMon [2]. Adotamos as seguintes convenções para tratar os valores dos atributos: os atributos podem ser nominais (discretos) e contínuos. Um atributo é nominal ou discreto quando existir possibilidade da discretização de seus valores. Exemplo: o atributo SuspEntropy pode ter os seus valores definidos como tendo valor igual a 1, para quando o atributo analisado possui uma entropia suspeita e valor 0, para quando sua entropia for normal. Por outro lado, um atributo é considerado continuo quando esta discretização não for possível, como ocorre, por exemplo, com o atributo FileSize, que identifica o tamanho do arquivo. Quando o valor de um atributo for desconhecido atribuímos o símbolo "?". Também desenvolvemos os scripts de submissão simultânea dos artefatos aos sistemas de análise estática e dinâmica descritos. Criamos um programa, preparadados.c, que seleciona, a partir dos relatórios gerados pelas análises, os dados de interesse, unificando-os em um vetor de dados que é utilizado na classificação automática dos malwares pelo aprendizado de máquina. Para o aprendizado de máquina, preparamos e configuramos o Framework de Aprendizado de Máquina (FAMA), desenvolvido no Instituto Militar de Engenharia [21]. Foram utilizados os algoritmos de aprendizado de máquina C5.0 [19] e o Random Forest [20] na classificação automática.

A seguinte lista resume as informações obtidas no processo de análise estática. Estes atributos foram escolhidos a partir de trabalhos [3], [4], [11], [12], [13], [14], [20] realizados que abordam características observadas em arquivos e em seus códigos e que podem auxiliar na identificação de um arquivo como suspeito.

- Yara: obtido pela execução do aplicativo yara.py, integrado ao Cuckoo Sandbox, e que permite verificar *malwares* que já são conhecidos e que já estão mapeados;
- EPEiD: verifica se existe alguma assinatura de código de cifradores, empacotadores e compiladores conhecidos e que foram utilizados na criação do código do *malware*.
- DateSusp: corresponde ao atributo de identificação de datas suspeitas. Datas de criação suspeitas foram obtidas como resultado da análise utilizando o aplicativo pescanner.py. A data é considerada suspeita se o ano for menor que 2000 ou maior que o ano atual. O cálculo é feito com base no *timestamp* do arquivo e indica quando o compilador produziu o arquivo. Quando a data for suspeita é atribuído o valor 1 para o atributo e quando for normal é atribuído 0.

- EPSusp: permite identificar se o ponto de entrada de uma seção PE é suspeita. Um ponto de entrada de uma seção é o nome de uma seção PE que contém o AddressofEntryPoint. O AddressofEntryPoint para arquivos legítimos ou não empacotados normalmente reside em uma seção nomeada .code ou .text para programas em modo usuário e PAGE ou INIT para *drivers* de *kernel*. Uma seção PE é considerada suspeita se o endereço do ponto de entrada está localizado na última seção do arquivo PE ou se o endereço não é reconhecido como normal. Quando o ponto de entrada da seção for suspeito é atribuído o valor 1 para o atributo e quando for normal é atribuído 0.
- SuspEntropy: identifica seções que possuem uma entropia muito alta ou muito baixa. A entropia é um valor entre 0 e 8 e que identifica a aleatoriedade dos dados em um arquivo. Esta classificação deriva-se do trabalho de LYDA [11] que sugere que um arquivo com uma entropia maior que 7 deve ser considerado como suspeito. ZABIDI [12] utiliza a entropia entre 0 e 1 ou maior que 7 ($0 < y < 1$ ou $y > 7$) para identificar um arquivo suspeito. Por exemplo, um arquivo com uma sequência muito longa de caracteres iguais possui baixa entropia e dados criptografados ou compactados possuem alta entropia. O cálculo da entropia permite identificar seções que possuem código anormal ou empacotado. Quando a entropia for suspeita, ou seja, maior que sete ou menor que um, é atribuído o valor 1 para o atributo e quando for normal é atribuído 0. A entropia é calculada utilizando a equação [11]:

$$H(x) = \sum_{i=1}^{c} - p_i * \log_2 p_i \quad (1)$$

  onde, *p(i)* é a probabilidade da *i-ésima* unidade de informação em *x* séries de eventos de *n* símbolos. Quando existirem 256 possibilidades, então o valor da entropia variará entre 0 e 8.
- NumberOfSections: número de seções que compõem a imagem de um arquivo PE.
- SuspiciousString: identifica se no código desmontado existe uma *string* suspeita. Uma *string* é considerada suspeita se ela constar no índice de *strings* suspeitas que definimos de acordo com [13], [14]. Quando é encontrada uma *string* suspeita é atribuído o valor 1 para o atributo e quando não for encontrada é atribuído 0.
- FileSize: tamanho do arquivo. 97% dos *malwares* identificados nos últimos cinco anos possuem tamanho menor que 1MB [20].
- CRCSusp: caso o CRC calculado e o real sejam divergentes, o arquivo é considerado como suspeito. Quando é considerado suspeito, é atribuído o valor 1 e quando o arquivo não é considerado suspeito é atribuído o valor 0 para o atributo.

Para a análise dinâmica foram escolhidas as chamadas de APIs do Windows que constituem as características do vetor de dados da análise dinâmica. Utilizamos para a escolha destes atributos o seguinte conjunto catalogado por [14]. A execução combinada destes atributos permite identificar se um artefato pode ser considerado um *malware* quando submetido a um algoritmo de aprendizado de máquina [15], [22], [23]:

- CreateFile: cria um arquivo ou abre um arquivo existente. Pode ser utilizado por um *malware* para despejar um arquivo no sistema.
- CreateMutex: cria uma exclusão mútua de objeto que pode ser usada por um *malware*.
- CreateRemoteThread: utilizado para iniciar uma thread em um processo remoto. Um *malware* pode utilizá-lo para injetar código em um processo existente.
- CreateService: cria um serviço que pode ser iniciado em tempo de boot. Um *malware* pode utilizá-lo para persistência ou para carregar *drivers*.
- DeleteFile: exclui um arquivo.
- FindWindow: busca por uma janela aberta no *Desktop*.
- OpenMutex: abre um *handle* para um objeto com exclusão mútua. Um *malware* pode utilizar uma exclusão mútua para evitar a infecção de um sistema por diferentes instâncias de um mesmo malware. Exemplo: quando um *trojan* infecta um ambiente o primeiro passo é obter um *handle* para um *mutex* nomeado, se o processo falhar o processo do *malware* é encerrado.
- OpenSCManager: abre um *handle* para o gerenciador de controle de serviços. Isto permitirá ao *malware* interagir com os processos dos serviços do Windows, iniciando-os ou parando-os.
- ReadProcessMemory: usado para ler a memória de um processo remoto. Regiões válidas de memória podem ser copiadas utilizando esta API. Um *malware* pode fazer uso desta API para obter, por exemplo, o número do cartão de crédito de um usuário, utilizando um código de busca.
- RegDeleteKey: apaga uma chave de registro e subchaves.
- RegEnumKeyEx: enumera as subchaves de um registro aberto. Pode ser utilizado em conjunto com RegDeleteKey para apagar recursivamente chaves de registro.
- RegOpenKey: abre um *handle* para controlar a leitura e edição de um registro.
- ShellExecute: utilizado para executar um outro programa.
- URLDownloadToFile: faz um *download* de n bits da Internet e os salva em um arquivo.
- WriteFile: grava os dados no dispositivo de saída.
- WriteProcessMemory: grava dados em um processo remoto. Utilizado na injeção de código.
- ZwMapViewOfSection: mapeia a visão de uma seção em um espaço de endereçamento virtual.
- LoadDll: função de baixo nível que carrega uma DLL em um processo. Pode indicar que o programa atua de forma furtiva.
- GetProcedureAddress: recupera o endereço de uma função carregada na memória. Usada para importar funções de outras DLLs em adição às importadas no cabeçalho do arquivo PE.
- QueryValueKey: acessa os valores de uma chave de registro.

- IsDebuggerPresent: verifica se o processo está sendo depurado. Utilizado na detecção de *debuggers*.
- GetSystemMetrics: obtém informações de configurações do sistema.
- SetInformationFile: altera as informações de um arquivo.
- SetWindowsHookExA: define uma *hook function* que será chamada quando um evento ocorrer. Normalmente é utilizada por *keyloggers* e *spywares*.
- OpenMutant: abre um *handle* para o objeto para execução exclusiva da instância do *malware*.

O valor de cada um destes atributos é o número de vezes que cada uma das chamadas de API foi feita pelo artefato durante sua execução.

Para executar a análise dinâmica, montamos um ambiente isolado, para a coleta de informações dos *malwares*. Utilizamos o Cuckoo Sandbox para realizar a análise dinâmica.

V. UNIFICAÇÃO DA ANÁLISE DE MALWARES

A unificação das análises estática e dinâmica de *malwares* compreende as seguintes etapas que estão ilustradas na Fig. 2.

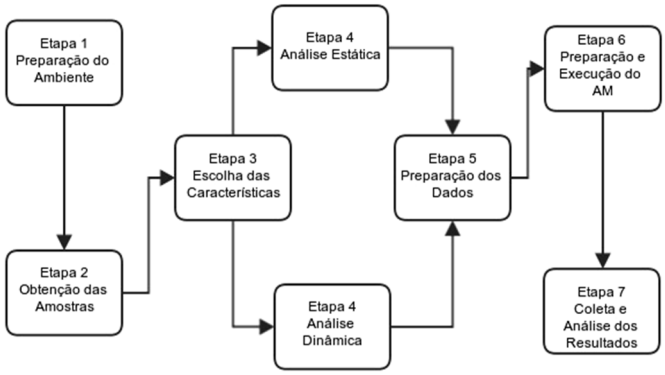

Figura 2. Etapas da unificação da análise de *malwares*.

Na primeira etapa foi preparado o ambiente de análise composto pela máquina hospedeira (host) utilizada para realizar o gerenciamento das tarefas de análise e hospedar a máquina virtual com o sistema alvo da atuação dos arquivos de *malwares* e não *malwares*.

Na etapa 2, foram obtidas as amostras de *malwares* e não-*malwares*. Os *malwares* foram obtidos no site VirusShare (http://vírusshare.com), totalizando 131.073 malwares de diversos formatos, tais como PE32, HTML, ico, gif, pdf, doc, xls, jpg, etc. Após a análise, foram selecionados da população somente aqueles que possuíam o formato Windows PE32. A coleção possui arquivos de *malwares* que foram obtidos até o primeiro bimestre de 2014.

Além do conjunto de *malwares*, também foram obtidas amostras de arquivos que classificamos como não *malwares* para fins de treinamento com a base de *malwares* e submissão das características das duas populações aos algoritmos de aprendizado de máquina. Foram obtidos 2.659 exemplares de não *malwares*. Estes arquivos foram obtidos dos sites sourceforge.net (http://www.sourceforge.net), oldapps (http://www.oldapps.com), de servidores de ftp anônimos (http://www.ftp-sites.org/) e de arquivos de computadores pessoais.

Na etapa 3, foram selecionadas as características para análise dos *malwares* e que correspondem às características apresentadas na seção anterior deste artigo. Além das escolhas das características, classificamos os arquivos como *malwares* e não *malwares* e também dentre os tipos de *malwares*, seguindo a definição dada pelo CERT.BR[1]: *trojan*, vírus, *worm*, *backdoor*, *rootkit*, *bot* e *spyware*. A classe "benigno" foi também utilizada e representa os arquivos que não são *malwares*.

Para classificar os arquivos em tipos de *malwares* utilizamos o site do Virustotal (HTTP://www.virustotal.com). Classificamos os artefatos identificados como *malwares* em classes, de acordo com a nomeação mais comum atribuída ao artefato pelos antivírus da base do Virustotal. Todos os arquivos de *malwares* e não *malwares* que foram obtidos na etapa 2, foram verificados no Virustotal e identificados por pelo menos um antivírus.

Na etapa 4, foram executadas as análises estática e dinâmica dos *malwares*. Para iniciar a análise foram executados *scripts* de submissão dos arquivos de *malwares* e não *malwares* para a análise do pescanner e do Cuckoo Sandbox.

Na etapa 5, as informações foram tratadas e consolidadas. Para a seleção dos dados, utilizamos um programa escrito em linguagem C, preparados.c, que lê os arquivos de entrada, referentes às análises estática e dinâmica e seleciona os atributos de interesse para a análise, conforme definido na etapa 3. Assim, são gerados dois conjuntos de dados. Um referente às informações coletadas pela análise estática e outro com as informações coletadas a partir da análise dinâmica. Estes dados são filtrados de tal forma a permitir a unificação.

Na etapa 6, foram instalados e configurados os aplicativos e algoritmos de aprendizado de máquina, para a classificação e predição dos exemplos. Foi utilizado nesta etapa o *framework* de aprendizado de máquina FAMA.

Na última etapa, os dados obtidos na etapa de aprendizado de máquina foram organizados e comparados. Os resultados foram obtidos utilizando o método de validação cruzada com 10 partições. As métricas que utilizamos na apuração dos resultados foram a acurácia, a precisão, a revocação e a média harmônica de precisão e revocação, ou F-1.

VI. EXPERIMENTOS

Foram realizados um total de nove experimentos. Os experimentos I, II e III comparam o desempenho da identificação de *malwares* utilizando a classificação binária. São considerados respectivamente, os atributos das abordagens dinâmica, estática e unificada. Os experimentos de IV até IX consideram o problema da classificação utilizando a categorização definida para tipos de *malwares*. Além destas, também utilizamos a categoria benigno, pois um artefato pode ser um não *malware*. A diferença entre as duas séries de experimentos, IV-VI e VII-IX é o uso de um atributo para analisar os artefatos. Na primeira série, a classificação multiclasses a priori é utilizada na classificação. Na segunda

série, a classe predita nos experimentos I, II e III, *malware* ou não *malware* é a utilizada como entrada para o processo de classificação.

Foram escolhidos aleatoriamente 6.292 exemplos a partir do conjunto de exemplos e suas quantidades e proporções estão descritas na tabela II.

TABELA II. EXPERIMENTOS DA ANÁLISE DE *MALWARES*.

Exp	Análise	CLASSE	A PRIORI	%
I, II,III	Unificada, Estática, Dinâmica	Maligno	3633	57,74%
		Benigno	2659	42,26%
IV – IX	Unificada, Estática, Dinâmica	Trojan	1941	30,85%
		Virus	585	9,30%
		Worm	747	11,87%
		Backdoor	111	1,76%
		Rootkit	97	1,54%
		Bot	47	0,75%
		Spyware	105	1,67%
		Benigno	2659	42,26%

O fator de confiança (CF) utilizado para testar a efetividade da pós-poda do algoritmo C5.0 foi igual a 0.35. O número de árvores do Random Forest foi definido como 100 e o número de atributos aleatórios utilizados na construção das árvores do comitê foi definido como 6. Todas estas configurações dos algoritmos de aprendizado de máquina foram ajustadas pelos primeiros experimentos e depois fixadas para todas as séries.

Todos os experimentos foram validados utilizando o método de validação cruzada com 10 partições e as métricas utilizadas para avaliar o desempenho das classificações foram a acurácia, a precisão, a revocação e o F-1 (medida-F).

VII. RESULTADOS

A tabela III resume os resultados de acurácia obtidos pelas análises utilizando classes binárias e múltiplas. Os melhores resultados estão destacados em negrito.

TABELA III. ACURÁCIA DOS EXPERIMENTOS.

EXP	ANALISE	Acurácia C5.0	Acurácia R Forest
I	UNIFICADA	91,91%	**95,75%**
II	ESTÁTICA	86,82%	**89,91%**
III	DINÂMICA	88,67%	**93,55%**
IV	UNIFICADA	82,96%	**88,94%**
V	ESTÁTICA	75,45%	**79,37%**
VI	DINÂMICA	80,13%	**86,52%**
VII	UNIFICADA	85,49%	**93,02%**
VIII	ESTÁTICA	77,80%	**82,53%**
IX	DINÂMICA	86,29%	**91,32%**

Podemos observar que os índices medidos de acurácia demonstram que os melhores resultados correspondem àqueles testes que utilizaram a análise unificada. A acurácia da análise para o problema de classificação binária foi igual a 91,91%, utilizando o classificador C5.0. Por sua vez, os índices obtidos com o classificador *Random Forest* foram superiores aos obtidos pelo classificador C5.0, obtendo índice igual a 95,75%.

O gráfico da Fig. 3 mostra que em todos os experimentos, os índices de acurácia do algoritmo *Random Forest* foram superiores aos obtidos pelo algoritmo C5.0.

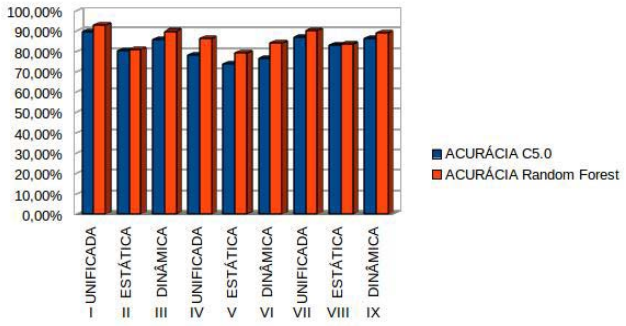

Figura 3. Resultados (Acurácia).

Para o problema de multiclassificação, o experimento VII obteve a melhor acurácia, igual a 93,02%, utilizando o *Random Forest*, e acurácia igual a 85,49% utilizando o C5.0. Também, pode-se observar que para estes experimentos o desempenho da análise unificada foi superior ao das análises isoladas.

Para melhor analisar os resultados dos experimentos IV até IX, utilizamos as métricas de precisão, revocação e F-1 para comparar os resultados das classes. Os melhores resultados foram obtidos pelo experimento VII e são apresentados na tabela IV.

TABELA IV. PRECISÃO, REVOCAÇÃO E MÉDIA HARMÔNICA DE PRECISÃO E REVOCAÇÃO (F-1) DO EXPERIMENTO VII.

CLASS	PREC C5.0	PREC RForest	REV C5.0	REV RForest	F-1 C5.0	F-1 RForest
BENIG	94,40%	99,90%	96,30%	99,90%	0,9534	0,9990
TROJ	78,90%	87,20%	82,50%	92,70%	0,8066	0,8987
VIRUS	70,60%	85,80%	60,30%	76,20%	0,6504	0,8072
WORM	88,30%	92,20%	80,20%	86,60%	0,8406	0,8931
BACKD	75,20%	95,50%	71,20%	75,70%	0,7315	0,8446
ROOTK	71,90%	82,00%	84,50%	93,80%	0,7769	0,8750
BOT	42,30%	78,80%	23,40%	55,30%	0,3013	0,6499
SPYW	72,90%	88,80%	89,50%	98,10%	0,8035	0,9322

A precisão do experimento VII permite identificar a capacidade dos algoritmos de aprendizado em classificar os casos positivos. Assim, verificamos que a precisão, por exemplo, para a classe Trojan foi igual a 87,20%, utilizando o algoritmo *Random Forest*, sendo superior a alcançada pelo algoritmo C5.0. A classe Trojan possui a maior proporção de exemplos classificados a priori, igual a 30,85%. Entretanto, o resultado da precisão desta classe, foi inferior ao das classes *worm*, *backdoor* e *spyware*. Cabe ressaltar aqui que a definição de classificação do CERT em relação a estes tipos, possui uma característica que deve ser destacada. O tipo Trojan é classificado em vários subtipos, como, Trojan Downloader, Trojan Dropper, Trojan Backdoor, Trojan Dos, Trojan Destrutivo, Trojan Clicker, Trojan Proxy, Trojan Spy e Trojan Banker. A classe Bot, que é uma classe definida pelo CERT.BR [1] e não por Szor [4], obteve o menor desempenho na classificação. O desempenho para a classificação em multiclasses foi bastante satisfatório. O F-1 se apresentou acima de 80% para todas as classes, com exceção da classe Bot.

VIII. CONCLUSÃO

Mostramos que a unificação das análises estática e dinâmica de *malwares*, com o emprego do aprendizado de máquina, pode ser utilizada como um método que auxilie o especialista a identificar com maior precisão e agilidade os códigos maliciosos. Aplicamos a classificação de duas maneiras: para a identificação do código de um artefato como malicioso ou não e de acordo com as definições de tipos de *malwares* derivadas do CERT.BR. Verificamos que em todos os cenários montados, a técnica de unificação das análises estática e dinâmica de *malwares*, apresentou melhor acurácia que as técnicas isoladas de análise estática e dinâmica. Os índices de acurácia obtidos para a análise unificada foram superiores a 90%, em ambas as abordagens de classificação, utilizando a classificação binária e a classificação multiclasses. Os métodos de classificação utilizados na classificação automática de *malwares* foram baseados em árvores de decisão, utilizando o algoritmo de aprendizado de máquina C5.0 e o método de comitê, utilizando o algoritmo *Random Forest*, implementados no *framework* de aprendizado FAMA. Os melhores resultados foram obtidos com o algoritmo *Random Forest*.

Além da acurácia, utilizamos o F-1 para medir o desempenho dos algoritmos. Estes índices mostram que houve um aumento no desempenho quando utilizamos a técnica unificada.

IX. SUGESTÕES DE TRABALHOS FUTUROS

Neste nosso trabalho utilizamos uma abordagem *batch* para a análise de *malwares*. Mas, sabemos que desenvolvedores de *malwares* aperfeiçoam diariamente suas técnicas para impedir que os especialistas e sistemas de segurança detectem suas ações. Buscam, sobretudo, explorar a capacidade de interligação dos sistemas via redes para disseminar e ampliar seus ataques. Uma abordagem *online*, que seja capaz de aprender um novo exemplo, de forma incremental, seria útil para mitigar a complexidade computacional dos métodos *batch* e realizar a predição *online* de códigos maliciosos.

Outras sugestões de trabalhos são a aplicação de nossa metodologia utilizando outros algoritmos de aprendizado de máquina como o AdaBoost e o SVM, a criação de comitês heterogêneos e a utilização de classificadores especializados por classes.

Também seria interessante a aplicação da abordagem unificada em outras plataformas que utilizem outros sistemas operacionais, como, por exemplo, Android, utilizar outras bases de *malwares* e aplicar o método em outros formatos de arquivos como pdf e documentos do Office.

A seleção de características pode ser melhorada e um método de seleção baseado no reconhecimento de padrões de sequências de símbolos no código desmontado pode ser útil.

A integração do processo de análise e de classificação em um único *framework* também pode ser uma solução para melhorar a classificação de *malwares*.

REFERÊNCIAS

[1] CERT.BR, Cartilha de segurana para internet, Centro de Estudos, Resposta e Tratamento de Incidentes de Segurana no Brasil, Brasil, Dec. 2013. [Online]. Available: http://cartilha.cert.br/malware

[2] D. Oktavianto and I. Muhardianto, Cuckoo Malware Analysis. Packt Publishing Ltd, 2013.

[3] C. H.Malin, E. Casey, J. M. Aquilina, and C. W. Rose, Malware Forensics Field Guide For Windows Systems. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier, 2012.

[4] P. Szor, The art of computer virus research and defense. Pearson Education, 2005.

[5] WILLEMS, C., HOLZ, T. e FREELING, F. Toward automated dynamic malware analysis using cwsandbox. IEEE Security and Privacy, 5(2):32–39, 2007.

[6] BRANCO, R. R., BARBOSA, G. N. e 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.

[7] C. Eagle, The IDA pro book: the unofficial guide to the world's most popular disassembler. No Starch Press, 2008.

[8] "Pyew. a python tool for static malware analysis," 2014. [Online].

[9] Available: https://code.google.com/p/pyew/.

[10] S. Arshad and M. Ahmad, "Speartrack-antimalware," Tech. Rep., 2014.

[11] P. Harvei, "Exiftool by phil harvey. read, write and edit meta information" 2014. [Online]. Available: http://www.sno.phy.queensu.ca/ phil/exiftool/

[12] LYDA, R. e HAMROCK, J. Using entropy analysis to find encrypted and packed malware. IEEE Security & Privacy, 5(2):40–45, 2007.

[13] ZABIDI, M. N. A., MAAROF, M. A. e ZAINAL, A. Malware analysis with multiple features. Em Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on, págs. 231–235. IEEE, 2012

[14] HEXACORN. Hexdive 0.4, 2014. URL http://www.hexacorn.com/blog/2012/08/.

[15] SIKORSKI, M. e HONIG, A. Pratical Malware Analysis. William Pollock, San Francisco, CA, 2012.

[16] DE ANDRADE, C. A. B., MELLO, C. G. e DUARTE, J. C. Malware automatic analysis. Computational Intelligence and 11th Brazilian Congress on Computational Intelligence (BRICS-CCI & CBIC), págs. 681–686, 2013.

[17] QUINLAN, J. Induction of decision trees. Machine Learning, 1(1):81–106, 1986.

[18] QUINLAN, J. C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[19] QUINLAN, R. See5/c5.0. rulequest research: datamining tools, 2010. URL http://www.rulequest.com.

[20] BREIMAN, L. Random forests. Machine learning, 45(1):5–32, 2001.

[21] FAMA. Framework de aprendizado de máquina, 2014. URL: https://code.google.com/p/fama/.

[22] FORTINET. Understanding how file size affects malware detection, 2014. URL http://www.fortinet.com/sites/default/files/whitepapers /MalwareFileSize.pdf.

[23] MOHAISEN, A., WEST, A. G., MANKIN, A. e ALRAWI, O. Chatter: Exploring classification of malware based on the order of events. 2014.

[24] SAMI, A., YADEGARI, B., RAHIMI, H., PEIRAVIAN, N., HASHEMI, S. e HAMZE, A. Malware detection based on mining api calls. Em Proceedings of the 2010 ACM Symposium on Applied Co

[25] LIGH, M., ADAIR, S., HARTSTEIN, B. e RICHARD, M. Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code. Wiley Publishing, 2010.mputing, págs. 1020–1025. ACM, 2010.

[26] KOLTER, Jeremy Z.; MALOOF, Marcus A. Learning to detect malicious executables in the wild. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004. p. 470-478.

[27] RAMAN, Karthik. Selecting features to classify malware. InfoSec Southwest, v. 2012, 2012.

[28] KONG, Deguang; YAN, Guanhua. Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013. p. 1357-1365.

Reinaldo José Mangialardo é graduado em Processamento de Dados pelo Centro Universitário UniFIEO, Osasco, São Paulo, Brasil, em 1993. Suas áreas de interesse de pesquisa atuais são a análise de malwares, defesa cibernética, inteligência artificial, integridade e disponibilidade de redes e sistemas, segurança em computação em nuvem, descoberta de vulnerabilidades, sistemas operacionais e engenharia reversa.

Julio Cesar Duarte Recebeu o título de Doutor pela Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, em 2009. Atualmente é professor do Instituto Militar de Engenharia (IME), Rio de Janeiro, Brasil. Suas áreas de interesse e pesquisa são multidisciplinares com ênfase no desenvolvimento de sistemas nas seguintes áreas: aprendizado de máquina, defesa cibernética, processamento em linguagem natural do português e análise de malwares.