# Agrupamento de malware por comportamento de execução usando lógica fuzzy

**Lindeberg Leite**[1], **Daniel G. Silva**[2], **André Grégio**[3]

[1] Departamento de Engenharia Elétrica – Universidade de Brasília
Brasília, DF / Polícia Federal

[2]Departamento de Engenharia Elétrica – Universidade de Brasília
Brasília, DF.

[3]Departamento de Informática – Universidade Federal do Paraná
Curitiba, PR.

lindeberg.lpl@dpf.gov.br, danielgs@ene.unb.br, gregio@inf.ufpr.br

***Abstract.*** *The threat of malware variants continuously increases. Several approaches have been applied to malware clustering for a better understanding on how to characterize families. Among them, behavioral analysis is one that can use supervised or unsupervised learning methods. This type of analysis is mainly based on conventional (crisp) logic, in which a particular sample must belong only to one malware family. In this work, we propose a behavioral clustering approach using fuzzy logic, which assigns a relevance degree to each sample and consequently enables it to be part of more than one family. This approach enables to check other behaviors of the samples, not visualized in conventional logic. We compare the chosen fuzzy logic algorithm — Fuzzy C-Means (FCM) — with K-Means so as to analyze their similarities and show the advantages of FCM for malware behavioral analysis.*

***Resumo.*** *A ameaça de variantes de malware aumenta continuamente. Várias abordagens para agrupamento de malware já foram aplicadas para entender melhor como caracterizar suas famílias. Destas, a análise comportamental pode usar tanto métodos de aprendizado supervisionado como não-supervisionado. Neste caso, a análise é comumente baseada em lógica convencional, onde um dado exemplar deve pertencer a apenas uma família. Neste trabalho, propõe-se uma abordagem de agrupamento comportamental por lógica fuzzy, que atribui um grau de relevância à cada exemplar e permite que este pertença a mais de uma família. Essa abordagem possibilita verificar outros comportamentos das amostras, não visualizados na lógica convencional. Compara-se o algoritmo escolhido — Fuzzy C-Means (FCM) — com o algoritmo K-Means para analisar similaridades e mostrar as vantagens do FCM na análise comportamental de malware.*

## 1. Introdução

A quantidade de *malware* vem aumentando significativamente, ano após ano. De acordo com a McAfee [Intel Security 2014], mais de 307 novas ameaças surgem a cada minuto,

 

ou seja, mais de cinco por segundo. No terceiro trimestre de 2014, já eram contabilizados mais de 40 milhões de novos tipos de *malware*. Esses exemplares, embora considerados "novos" ou "únicos", nada mais são do que mutações de *malwares* já existentes. Pequenas variações no código, como mudanças em uma lista de endereços IP para varredura ou comunicação, em *strings* ou o uso de mecanismos de ofuscação (*packers*) fazem com que o exemplar seja considerado novo em relação aos demais. Os exemplares decorrentes de mutação de outros são conhecidos como *variantes*. As variantes, responsáveis pela elevação na quantidade de *malware* em atuação, dificultam a criação de assinaturas de detecção e facilitam a tarefa dos atacantes. Por outro lado, estimula-se a busca por técnicas capazes de detectar, classificar e/ou agrupar tais exemplares de maneira eficiente e eficaz.

Os fabricantes de antivírus (AV) criam assinaturas estáticas usando procedimentos majoritariamente manuais, pois é preciso que um analista humano descompile ou desmonte o código a fim de encontrar um padrão que, ao mesmo tempo em que detecte um *malware*, não gere falsos-positivos que prejudiquem a experiência do usuário. Devido a isso, grande parte dos exemplares de *malware*, incluindo os mais complexos, permanece não detectada por um tempo além do adequado. Por exemplo, estima-se um típico intervalo de 54 dias entre a disseminação de um dado *malware* e sua detecção por algum AV; 15% das amostras passam desapercebidas por 180 dias [Damballa 2009]. Por isso, torna-se essencial desenvolver métodos automatizados para detectar e agrupar exemplares de *malware* desconhecidos, ou seja, que ainda não pertencem aos bancos de dados de assinaturas dos AV [Salehi et al. 2012].

A dissecação de um exemplar de *malware* para melhor compreensão de seu modo de atuação (disseminação, infecção, ocultação de rastros, comunicação com o atacante, entre outras ações) envolve formas distintas de análise: (*i*) a análise estática atua diretamente no binário sem a necessidade de execução, ou seja, obtém-se informações do cabeçalho, verifica-se pela presença de *packers*, descompila-se, quando possível, para se avaliar suas instruções e se tentar identificar blocos suspeitos, busca-se por *strings* que correspondam a endereços IP, URLs, e-mail, etc.; (*ii*) a análise dinâmica consiste na observação da execução do *malware* em um ambiente controlado (*sandbox*) a fim de se extrair seu comportamento, isto é, quais são as atividades realizadas no nível do sistema operacional e da rede que podem ser utilizadas para definir como a infecção ocorre; (*iii*) a combinação de ambos os tipos, a fim de se valer das vantagens providas tanto pela análise estática quanto pela dinâmica.

Além da análise estática, dinâmica ou mista para detecção, há uma preocupação crescente com o agrupamento de exemplares na família correta. Com isso, entende-se melhor as características de cada grupo e aumenta-se a chance de detecção. Ademais, descobrir o rótulo adequado de um determinado artefato é fundamental para conhecer antecipadamente sua intenção maliciosa, bem como seu método de infecção e propagação. Em uma abordagem investigativa, saber o rótulo correto direciona o esforço do analista para as atividades maliciosas da amostra, inclusive para antever contramedidas. Nesse sentido, técnicas de aprendizado de máquina podem ser aplicadas para se realizar o processo de agrupamento (em inglês, *clustering*) de forma automatizada. Tradicionalmente, tais tentativas envolvem a aplicação de algoritmos fundamentados na lógica tradicional ou *"crisp"*, a qual se associa à teoria clássica de conjuntos: sob esta ótica, um exemplar de *malware* pertence exclusivamente a uma família, dentre todas as possíveis. No entanto,

 

com a sofisticação dos processos de criação de *malware* usando kits de faça-você-mesmo, os exemplares produzidos passaram a empregar intensivamente o reuso de código. Dessa forma, o *malware* atual não é mais limitado a exibir o comportamento específico de uma classe pré-determinada, podendo carregar ao mesmo tempo características de várias classes. Logo, faz-se necessário um método capaz de retratar, de forma mais fidedigna, a dinâmica comportamental de um *malware*. Neste artigo, propõe-se um método para agrupamento de exemplares de *malware* pelo seu comportamento de execução, o qual tem sua base na lógica fuzzy. Realizou-se um estudo comparativo entre um método que emprega a lógica *crisp* — *K-Means* — e outro que adota a lógica fuzzy — *Fuzzy C-Means* (*FCM*) [Bezdek et al. 1984]. Este trabalho, portanto, fornece as seguintes contribuições:

- Aplicação de um novo método para agrupar exemplares de *malware* de forma mais flexível que os classificadores tradicionalmente utilizados;
- Um processo de rotulação eficaz e realista, pois considera os diversos comportamentos exibidos por um *malware* moderno. Nos casos em que há *clusters* sem rótulos, o *FCM* apresentou vantagens na atribuição de um nome ao grupo devido à sua matriz de pertinência;
- Avaliação comparativa entre os algoritmos de agrupamento *K-Means* (lógica tradicional) e *FCM* (lógica *fuzzy*), permitindo a análise dos resultados para verificação de semelhanças entre eles, bem como a análise do tempo de execução gasto por algoritmos de diferentes paradigmas aplicados ao mesmo *dataset*. Constatou-se um alto grau de equivalência entre os *clusters* produzidos por ambos e mensurou-se o tempo de execução dos experimentos realizados com eles.

O restante deste artigo está organizado da seguinte forma: a Seção 2 discute os trabalhos relacionados; a Seção 3 apresenta as etapas da análise comportamental de *malware* e a extração de características para posterior agrupamento; a Seção 4 introduz o processo de aprendizado de máquina e os algoritmos usados; a Seção 5 mostra o resultado dos experimentos realizados e, por fim, a Seção 6 expõe as considerações finais e os trabalhos futuros.

## 2. Trabalhos Relacionados

Nesta seção, são apresentados alguns trabalhos correlatos que envolvem a aplicação de algoritmos de aprendizado de máquina na classificação ou agrupamento de exemplares de *malware*.

[Salehi et al. 2012] criaram um método para detecção de *malware* baseado nas chamadas de função de API e seus argumentos. Primeiramente, efetuou-se o processo de coleta de exemplares, os quais foram divididos em dois conjuntos (*malware* e não *malware*), nos quais 385 deles eram programas nativos encontrados no sistema operacional Windows ou outras ferramentas consideradas benignas e 826 eram programas considerados maliciosos. Esses arquivos foram obtidos de [Sami et al. 2010]. Os binários executaram em um sistema operacional Windows virtualizado no VMware e o comportamento foi monitorado pelo `WINAPIOverride32`. Foram monitoradas 126 funções de API das seis DLLs (*Dynamic-Link Library*) mais importantes do Windows: `advapi32.dll`, `kernel32.dll`, `ntdll.dll`, `user32.dll`, `wininet.dll` e `ws2_32.dll`. Além da monitoração das funções de API, foram observados também seus argumentos. O resultado dessa monitoração foi convertido para o formato ARFF e,

 

no processo de classificação, a melhor acurácia (97%) foi obtida com o algoritmo *Functional Trees* (FT) do WEKA.

[Mangialardo 2015] elaborou uma técnica integrada de análises estática e dinâmica na identificação de *malware* utilizando aprendizado de máquina. O processo iniciou-se com a coleta de exemplares, então divididos em dois conjuntos: um composto por 2.659 executáveis benignos obtidos do *SourceForge*[1], *OldApps*[2] e outras fontes; outro por 131.073 exemplares de *malware* obtidos do *VirusShare*[3]. No processo de escolha das características, levou-se em consideração tanto *strings* extraídas da análise estática como funções de API capturadas da análise dinâmica. Para formar o dicionário de *strings* da análise estática, usaram-se várias ferramentas ou informações presentes no cabeçalho dos binários, tais como *Yara*[4], *PEiD*[5] e dados obtidos pelo *PEScanner*[6] (DateSusp, EPSusp, SuspEntropy, NumberOfSections, IATSusp, SuspiciousString, FileSize, CRCSusp). Esse dicionário posssui os atributos mais utilizados por *malware*. No caso da análise dinâmica, montou-se um dicionário de termos semelhante ao visto em [Andrade et al. 2013]. Para o processo de aprendizado, foi utilizado o framework FAMA [Duarte et al. 2012]. Os algoritmos de aprendizado de máquina escolhidos para a tarefa de classificação foram o C5.0 (árvore de decisão) e a técnica *Random Forest*, cuja acurácia foi de 95,75%.

[Provataki and Katos 2013] apresentaram um *framework* para análise forense de *malware* baseado em Cuckoo Sandbox, usado para avaliar e elaborar relatórios sobre o comportamento dos exemplares executados. O trabalho abordou as limitações da análise dinâmica e ampliou a funcionalidade da ferramenta Cuckoo Sandbox a fim de automatizar o processo de correlacionamento e investigação de várias execuções de um binário suspeito sobre plataformas distintas de sistemas operacionais. A abordagem apresentada permite que o analista identifique mudanças comportamentais quando o artefato é executado e possibilita responder questões relacionadas às atividades do programa malicioso que auxiliem uma investigação mais aprofundada.

[Firdausi et al. 2010] demonstraram uma prova de conceito de um método para detecção de *malware*. Inicialmente, o comportamento dos artefatos foi obtido via análise dinâmica no sistema Anubis [Iseclab 2015], e os relatórios gerados foram pré-processados de forma a se criar vetores para classificação. Procedeu-se uma comparação de desempenho de cinco diferentes técnicas — *K-Nearest Neighbors* (KNN), *Naive Bayes*, Árvore de Decisão (J.48), *Support Vector Machine* (SVM) e Rede Neural do tipo Perceptron de Múltiplas Camadas (MLP). Foi produzido um pequeno conjunto de amostras maliciosas e dados de executáveis benignos. Os resultados obtidos mostraram que, em geral, o melhor desempenho é conseguido pela árvore de decisão J.48 com uma taxa de falsos-positivos de 2,4% e acurácia de 96,8%.

[Pirscoveanu 2015] investigou as inconsistências associadas à geração de rótulos por mecanismos antivírus no processo de aprendizado não-supervisionado, com objetivo de realizar análise comportamental de *malwares*. Uma versão personalizada do Cuckoo

[1] `https://sourceforge.net`
[2] `http://www.oldapps.com`
[3] `https://virusshare.com`
[4] `http://virustotal.github.io/yara/`
[5] `https://www.aldeid.com/wiki/PEiD`
[6] `https://code.google.com/archive/p/malwarecookbook/`

 

Sandbox foi utilizada para coletar ações de cerca de 270.000 exemplares maliciosos e, posteriormente, criar um dicionário de termos consistindo de chamadas de API que foram executadas com sucesso e aquelas que falharam, com seus respectivos códigos de retorno. Além disso, avaliou-se os resultados de detecção fornecidos por fabricantes de antivírus. Com isso, criou-se uma solução que estabelece como rótulo o nome de família que recebeu a maioria do votos dos AVs. Devido às inconsistências entre os rótulo de AVs, a distância de Levensthein foi usada a fim de medir a similaridades entre os rótulos.

[Huang et al. 2013] explora a possibilidade de utilizar técnicas *fuzzy* para identificar semelhanças entre famílias de softwares maliciosos. No modelo proposto, características maliciosas de um programa são inicialmente captadas pelo reconhecimento de padrão *fuzzy* e, em seguida, uma ontologia *fuzzy* para análise comportamental de *malwares* é apresentada. A abordagem é composta por uma inferência *fuzzy* e um mecanismo semântico de tomada de decisão para descrever as mudanças do valor de chave de registo, conexão de rede e alteração de arquivos, permitindo construir a ontologia e regras de comportamento. O modelo é capaz de detectar programas maliciosos desconhecidos e variantes de *malwares*.

Para fins de avaliação dos resultados deste trabalho frente ao paradigma da lógica convencional, também se realizou uma análise comparativa entre o algoritmo de agrupamento *K-Means*, bastante empregado na literatura, e o algoritmo *FCM*.

## 3. Análise Comportamental

Nesta seção, discute-se brevemente as fases necessárias para a geração do vetor de características a ser usado na etapa de agrupamento com os algoritmos de aprendizado de máquina, bem como o ambiente de execução utilizado. Basicamente, todo o processo para se completar a análise comportamental é composto por três fases:

1. Coleta de exemplares de *malware*;
2. Captura das chamadas de funções das APIs;
3. Geração do dicionário de termos.

### 3.1. Ambiente de execução

As amostras foram processadas no sistema de análise dinâmica *Cuckoo Sandbox* executando no sistema operacional *Linux Ubuntu*. No *Cuckoo*, configuraram-se 3 máquinas virtuais com *Windows XP* atualizados com *Service Pack* 3. Além disso, para se aproximar de um ambiente real de usuário, *softwares* comumente usados como *Acrobat Reader, Skype, Flash, Office*, entre outros foram instalados nas máquinas virtuais. Esses programas foram baixados e instalados por meio do *Ninite* [Swieskowski and Kuzins 2016]

### 3.2. Coleta de *malware*

Nesta fase, foram obtidas amostras de *malware* do *site VirusShare*. Neste site, é possível a busca de exemplares para *download* por ano de recebimento. Assim, foram separados, de forma aleatória, exemplares de 2012, 2013, 2014, 2015 e 2016, totalizando a quantidade de 21.455 amostras consideradas maliciosas.

 

### 3.3. Captura das funções de API

Os 21.455 exemplares de *malware* coletados foram submetidos para execução no sistema de análise dinâmica *Cuckoo Sandbox*. Durante cada execução, esse analisador monitorou as funções de API chamadas pelo artefato sob análise e gerou um relatório em formato JSON para cada amostra. Os relatórios contêm informações do *malware* obtidas de maneira estática e dinâmica pelo Cuckoo.

### 3.4. Geração do dicionário de termos

Os relatórios gerados anteriormente passaram por um processo de filtragem (implementado por meio de um *script* em Python), usado para identificar as funções de API mais relevantes. Assim, para cada arquivo JSON gerado, levantou-se quais chamadas a funções de API são consideradas maliciosas. Para isso, as funções de API presentes nos relatórios gerados foram comparadas com 153 funções de API comumente utilizadas por *malware* de acordo com o trabalho de [Pirscoveanu 2015]. Apenas as funções acessadas pelo exemplar que pertencem a essa lista de 153 chamadas de funções foram utilizadas para a geração do dicionário de termos. Assim, após a filtragem e comparação de todos os 21.455 relatórios produzidos a partir da execução dos exemplares do conjunto de dados, obteve-se um dicionário composto por 144 termos (ou chamadas de funções).

O *script* de filtragem, após criar o dicionário de termos, gera um arquivo em formato CSV (*comma-separated values*). Este arquivo contém a quantidade de vezes em que uma dada função de API presente no dicionário de termos é chamada por cada um dos exemplares.

## 4. Aprendizado de Máquina

Nesta seção, apresenta-se a aplicação dos algoritmos de aprendizado de máquina nos dados obtidos na etapa de análise comportamental. O arquivo CSV gerado com os vetores de características extraídos a partir de todos os exemplares executados serviu de entrada tanto para o algoritmo *K-Means* quanto para o *FCM*. Ambos foram implementados em linguagem "R". Porém, devido à grande dimensão do vetor de características (144 termos) construído na etapa anterior, antes da aplicação dos algoritmos de agrupamento é necessário um procedimento para redução de dimensionalidade, cuja descrição vem a seguir.

### 4.1. Análise de Componentes Principais

Observe que o *dataset* inicial, gerado na etapa descrita na Seção 3, é composto por uma matriz de 144 colunas (dimensões) por 21.455 linhas (observações). Para viabilizar o tempo de execução dos algoritmos de agrupamento e reduzir o risco de baixo desempenho devido à "maldição da dimensionalidade" [Duda et al. 2001], é necessário proceder com algum pré-processamento que diminua a dimensão do problema.

Dessa maneira, aplicou-se a reconhecida técnica de Análise de Componentes Principais (PCA, do inglês *Principal Component Analysis*), que consiste em identificar uma projeção linear da matriz de dados $\mathbf{S}$ original

$$\mathbf{X} = \mathbf{AS} \tag{1}$$

 

tal que a nova matriz $\mathbf{X}$ possua um número de características menor que $\mathbf{S}$, simultaneamente atendendo à restrição de tais características serem descorrelacionadas entre si e de mínimo erro quadrático médio com respeito aos dados originais [Duda et al. 2001].

Consequentemente, o uso das características geradas, via PCA, reduz o número de colunas presentes na nova matriz de dados e, devido à restrição de descorrelação, também reduz o grau de redundância. Nesse sentido, é possível observar na Tabela 1 o acúmulo da variação dos dados (variância ou nível de energia) à medida que se considera mais componentes principais.

**Tabela 1. Acúmulo da variação dos dados**

| **PCA** | PC1 | PC2 | PC3 | PC4 | PC5 | PC6 | PC7 | PC8 |
|---|---|---|---|---|---|---|---|---|
| **Acumulada** | 49,6% | 60,2% | 68,4% | 73,2% | 77,0% | 79,8% | 82,0% | 84,0% |
| **PCA** | PC9 | PC10 | PC11 | PC12 | PC13 | PC14 | PC15 | PC16 |
| **Acumulada** | 85,8% | 87,1% | 88,2% | 89,2% | 90,1% | 90,9% | 91,7% | 92,4% |
| **PCA** | PC17 | PC18 | PC19 | PC20 | | | | |
| **Acumulada** | 93,2% | 93,8% | 94,4% | 94,8% | | | | |

Dado que as 20 primeiras componentes principais explicam 94,8% da variância dos dados, elas foram selecionadas como características do vetor de entrada tanto para o algoritmo *K-Means* quanto para o *FCM*. Vale destacar que essas 20 primeiras componentes principais são combinações lineares das 144 originais.

### 4.2. Métodos não-supervisionados

A ideia de algoritmos não-supervisionados é fornecer classificação de informações de acordo com os próprios dados, baseado em análises e comparações entre os seus valores numéricos. Dessa maneira, esses métodos classificam os dados automaticamente, sem a necessidade de supervisão [Borges 2010].

Entre esses algoritmos, o *K-Means* [MacQueen 1967] é amplamente utilizado. Tornou-se popular devido à sua simplicidade e tem sido, com sucesso, aplicado durante os últimos anos, sobretudo para agrupamento de *malwares*. O objetivo desse método é particionar as amostras em K grupos, minimizando a distância *intra-clusters* e maximizando a distância *inter-clusters* [Borges 2010]. A distância euclidiana é um critério de similaridade amplamente utilizado para as amostras no espaço euclidiano.

O *Fuzzy C-Means* proposto por [Dunn 1974] e estendido por [Bezdek 1981] é um método de agrupamento que permite estabelecer, para um certo dado, um determinado grau de relação com cada um dos agrupamentos obtidos. Ele possui funcionamento e estrutura semelhantes ao *K-Means*, mas possui uma abordagem *soft*, ao permitir que um dado não esteja associado exatamente com um único *cluster*. O grau de relação (também chamado de grau de pertinência) entre uma amostra e um agrupamento é um valor que está no intervalo [0, 1]. Uma pertinência próxima a 1, significa que o exemplar e o agrupamento em questão são similares. Caso contrário, se esse valor se aproxima de zero, indica dissimilaridade entre a amostra e o agrupamento analisado [Borges 2010].

### 4.3. Número de *clusters*

Uma das dificuldades em se aplicar os principais algoritmos de agrupamento, como o *K-Means* e o *FCM*, é determinar antecipadamente o número de *clusters* que deve ser gerado

 

a partir das amostras de entrada. Em alguns casos, como o do presente trabalho, não se sabe exatamente o número correto de *clusters*, pois quer-se agrupar automaticamente os exemplares de acordo com o comportamento de execução apresentado (e não por classes geradas por rótulos de AV). Desse modo, para se chegar a um valor de parâmetro apropriado, foram usados dois métodos de validação: o método *Elbow* e o *Silhouette*.

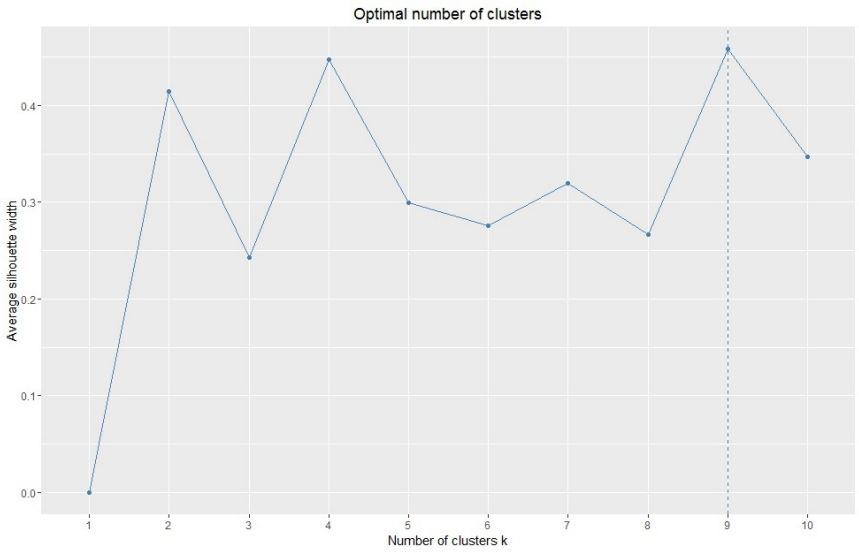


**Figura 1. Aplicação do método Silhouette nas amostras de entrada**

O método *Elbow* baseia-se na análise de variância em função do número de *clusters*. Tal como muitos outros métodos, ele exige o uso de um algoritmo de agrupamento, como *K-Means*, para executar o agrupamento e retornar a quantidade de variância produzida por cada *cluster* [Thorndike 1953]. O método Silhouette (Figura 1) refere-se a um método de interpretação e validação de consistência dentro dos *clusters*. A técnica fornece uma representação gráfica sucinta de como cada objeto se encontra dentro de seu *cluster*. O valor *silhouette* mede o quanto um objeto é similar ao seu próprio grupo (coesão) em comparação com os outros grupos (separação) [Rousseeuw 1987].

Com o método *Elbow*, não foi possível observar claramente o número de *clusters* necessários/adequados. Entretanto, a aplicação do método Silhouette resultou na sugestão de definição de 9 *clusters*, conforme Figura 1. Logo, primeiramente executou-se o algoritmo *K-Means* com $k = 9$, obtendo-se a distribuição das amostras conforme a Tabela 2.

**Tabela 2. Distribuição das amostras entre os 9 *clusters*, após aplicação do *K-Means*.**

| **Cluster** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **# Exemplares** | 5579 | 3526 | 740 | 1305 | 3656 | 409 | 1284 | 4765 | 191 |

Uma vez que o intuito é comparar o resultado dos algoritmos, o *FCM* também foi aplicado com a definição de 9 *clusters* como parâmetro de entrada. No caso do *FCM*, essa amostra não pertence completamente a um determinado cluster, os graus de pertinência indicam participação dela em cada cluster. Dessa forma, verificou-se em qual cluster o artefato possui maior pertinência, para fins de comparação com o *K-Means*. A distribuição dos exemplares resultante pode ser observada na Tabela 3.

 

**Tabela 3. Distribuição das amostras entre os 9 *clusters*, após aplicação do *FCM*.**

| **Cluster** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **# Exemplares** | 1945 | 2935 | 1830 | 399 | 1197 | 774 | 2277 | 5622 | 4476 |

## 5. Resultados dos Experimentos

Os resultados obtidos por meio da aplicação dos dois algoritmos de aprendizado de máquina foram analisados sob três aspectos: semelhança entre *clusters* produzidos, processo de rotulação e tempo de execução. A seguir, cada um dos aspectos supracitados é discutido.

### 5.1. Semelhança entre *clusters*

Para a análise deste aspecto, foi desenvolvido um *script* em *Python* com a finalidade de comparar os *clusters* gerados como saída de cada algoritmo aplicado. Conforme mostrado na Tabela 4, pode-se observar o grau de coincidência dos exemplares para cada par de *clusters*, medido em porcentagem. Cinco *clusters* das duas técnicas apresentaram um elevado grau de semelhança, acima de 90%: por exemplo, o *cluster* K3 e C6 mostraram semelhança de 98%. O *cluster* 2 do *K-Means* (K2) mostrou a maior diferença, ao alocar 48% dos seus dados no *cluster* 1 do *FCM* (C1) e 45% no *cluster* 2 do *FCM* (C2). Com base nesses resultados, pode-se concluir que os dois métodos produziram agrupamentos semelhantes, conforme Tabela 4.

**Tabela 4. Semelhança entre *clusters* produzidos por *K-Means* (K) e *FCM* (C).**

| **Clusters** | K1/C7 | K1/C8 | K2/C1 | K2/C2 | K3/C6 | K4/C5 |
|---|---|---|---|---|---|---|
| **Semelhança** | 40% | 60% | 48% | 45% | 98% | 91% |
| **Clusters** | K5/C2 | K5/C8 | K6/C4 | K7/C3 | K8/C9 | K9/C3 |
| **Semelhança** | 37% | 63% | 95% | 92% | 93% | 55% |
| **Clusters** | K9/C1 | | | | | |
| **Semelhança** | 40% | | | | | |

Os métodos *K-Means* e *FCM* possuem os processos de inicialização, iteração e término bastante semelhantes. A diferença reside no emprego da função de pertinência por parte do método *FCM*. De fato, pode-se interpretar o *K-Means* como um caso especial do *FCM* para uma função de pertinência *crisp*, i.e. que retorna 1 se o ponto de dados é mais próximo do centroide ou 0, caso contrário. No caso do *FCM*, sabe-se que a função de pertinência pode gerar valores entre 0 e 1. Ao analisar os *clusters* que apresentaram diferenças na Tabela 4, pode-se levantar a hipótese de que categorias de artefatos de comportamento mais complexo, à luz da função de pertinência *fuzzy*, levam a uma associação "diversificada" de *clusters*, o que não se alinha com os resultados equivalentes do *K-Means*.

### 5.2. Processo de rotulação

No momento de execução das amostras, o *Cuckoo* consulta a plataforma *online* do *VirusTotal*[7] de modo a verificar se o *malware* em questão já foi previamente enviado para

[7] `https://www.virustotal.com`

 

detecção pelos antivírus disponíveis. Em caso positivo, o *VirusTotal* retorna um conjunto de rótulos, que corresponde ao resultado da varredura por vários AVs. Esse conjunto é armazenado como uma estrutura de dados no arquivo *JSON* referente ao relatório de cada exemplar. Como o cenário usual consiste em se obter rótulos distintos e/ou inconsistentes para um mesmo exemplar, elegeu-se o antivírus Avast para ser responsável pela atribuição de uma classe aos exemplares do conjunto deste experimento. A escolha do Avast deu-se pelo fato deste apresentar taxa de detecção elevada, de acordo com o VB100 [VirusBulletin 2016].

**Tabela 5. Exemplares de *malware* rotulados com Avast encontrados em *clusters* resultantes da aplicação do *K-Means* no *dataset*.**

| **Cluster** | K1 | K2 | K3 | K4 | K5 | K6 | K7 | K8 | K9 |
|---|---|---|---|---|---|---|---|---|---|
| Trojan | 2118 | 847 | 83 | 287 | 1025 | 80 | 401 | 2170 | 27 |
| Worm | 132 | 373 | 32 | 79 | 135 | 44 | 153 | 26 | 1 |
| Spyware | 243 | 182 | 15 | 45 | 98 | 9 | 20 | 130 | 0 |
| Rootkit | 72 | 13 | 7 | 5 | 27 | 4 | 10 | 7 | 0 |
| PuP | 258 | 627 | 437 | 384 | 432 | 99 | 53 | 341 | 70 |
| Genérico | 1707 | 833 | 93 | 299 | 836 | 49 | 436 | 680 | 55 |
| Sem rótulo | 1049 | 651 | 73 | 206 | 1103 | 124 | 211 | 1411 | 38 |

Observa-se que o Avast, ao rotular as 21.455 amostras, distribuiu-as em cinco tipos de *malware* com escopo de comportamento bem definido—*Trojan*, *Worm*, *Spyware*, *Rootkit* e *Potentially Unwanted Program (PUP)*. O rótulo ”Genérico”corresponde a nomes genéricos atribuídos pelo Avast (em geral detectados por procedimentos heurísticos), enquanto que ”Sem rótulo”refere-se aos artefatos que não foram identificados por este AV.

Ao analisar a Tabela 5, nota-se que 33% das amostram foram rotuladas como *Trojan* e 23% forma enquadradas como ”Genérico”, perfazendo um total de 55% de todas as amostras. Além disso, esses dois rótulos se espalharam por todos os *clusters*. Na verdade, os rótulos *Trojan* e ”Genérico”são usados por muitos fornecedores de AV como rótulos guarda-chuva [Mohaisen and Alrawi 2013], ou seja, cada um desses rótulos abrange mais de uma família de *malware*.

Nesta seção, apresenta-se uma solução para rotulação baseada em detecção por AV. No entanto, diferentemente do trabalho de [Pirscoveanu 2015], utiliza-se a tabela de pertinência do *FCM* para auxiliar no processo: inicialmente, realiza-se a rotulação do agrupamento resultante do *K-Means* e, após isso, rotula-se com o *FCM*.

O rótulo atribuído a cada *cluster* advém da classe mais representativa encontrada dentro dele, de acordo com os resultados obtidos pelo Avast. Assim, o *cluster* K8 recebeu o rótulo *Trojan*, uma vez que possui 2170 exemplares classificados como tal.

Seguindo essa lógica, tem-se esta atribuição de rótulos: K8 - *Trojan*; K2 - *PuP*; K1 - *Spyware*; K5 - *Rootkit*; e k7 - *Worm*

Nota-se que o tipo *Trojan* está presente em todos os agrupamentos e que cada rótulo só pode ser atribuído a um único *cluster*. Desse modo, por exemplo, ao atribuir o rótulo *Trojan* ao K8, este não pode ser concedido também ao K2, mesmo que neste *cluster* a quantidade desse tipo de artefato seja maioria. Nesse caso, opta-se pelo segundo

 

mais representativo, no caso *PuP*. Ademais, como exitem apenas cinco rótulos, quatro agrupamentos ficam sem rótulos (K3, K4, K6 e K9)

**Tabela 6. *Malwares* rotulados com Avast encontrados em *clusters* resultantes da aplicação do *FCM* no conjunto de exemplares.**

| **Cluster** | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 |
|---|---|---|---|---|---|---|---|---|---|
| Trojan | 393 | 928 | 463 | 78 | 263 | 89 | 874 | 1806 | 2144 |
| Worm | 303 | 88 | 174 | 45 | 70 | 37 | 65 | 171 | 22 |
| Spyware | 101 | 110 | 72 | 8 | 40 | 19 | 127 | 164 | 101 |
| Rootkit | 8 | 21 | 11 | 3 | 5 | 7 | 27 | 58 | 5 |
| PuP | 381 | 425 | 220 | 96 | 351 | 443 | 131 | 400 | 254 |
| Genérico | 360 | 755 | 584 | 47 | 281 | 101 | 571 | 1669 | 620 |
| Sem rótulo | 399 | 608 | 306 | 122 | 187 | 78 | 482 | 1354 | 1330 |

Para o método *FCM*, repete-se o mesmo processo, chegando aos seguintes rótulos para cada agrupamento, conforme Tabela 6: C9 = *Trojan*; C1 = *Worm*; C8 = *Spyware*; C6 = *PuP*; e C7 = *Rootkit*.

Os agrupamentos C2, C3, C4 e C5 estão sem rótulo. No entanto, neste caso, diferentemente do *K-Means*, pode-se tirar vantagem da tabela de pertinência. Para ilustrar essa ideia, foram selecionadas 10 amostras de C2 com suas respectivas pertinências, conforme a Tabela 7.

**Tabela 7. Tabela de pertinência de C2 com 10 exemplares, e seus dois *clusters* (C1 e C8) mais pertinentes/aderentes a ele.**

| **Amostras/Cluster** | C1 (Worm) | C2 | C8 (Spyware) |
|---|---|---|---|
| Exemplar 1 | 30% | 32% | 7,5% |
| Exemplar 2 | 10% | 7,5% | 7,6% |
| Exemplar 3 | 35% | 43% | 4,0% |
| Exemplar 4 | 20% | 67% | 4,1% |
| Exemplar 5 | 18% | 28% | 16% |
| Exemplar 6 | 1,8% | 53% | 42% |
| Exemplar 7 | 6,7% | 86% | 2,4% |
| Exemplar 8 | 14% | 27% | 26% |
| Exemplar 9 | 22% | 44% | 11% |
| Exemplar 10 | 5,3% | 66% | 22% |

Na Tabela 7, C1 e C8 foram selecionados na medida em que apresentaram o maior grau de pertinência/aderência ao C2. Por exemplo, analisando o Exemplar 5, nota-se que sua pertinência ao C2 é 28%, seguido de C1 (18%) e C8 (16%). Desse modo, sugere-se o rótulo "Worm-Spyware"para o *cluster* C2. Repetindo esse procedimento para os três agrupamentos restantes sem rótulos, têm-se: C3 = *Worm-Trojan*; C4 = *PuP-Worm-Trojan*; e C5 = *PuP-Worm*

Por fim, vale destacar que a rotulação, de um modo geral, fornece um subsídio importante ao analista de *malwares*, ao especificar como rótulo um nome descritivo das características dos artefatos que ali se encontram. Em particular, o *FCM* permite verificar

 

graus de relacionamento entre os *clusters*, ajudando a observar outros comportamentos da amostra. Por exemplo, ao rotular C2 com *"Worm-Spyware"*, o *FCM* informa ao analista que atividades maliciosas de *Worms* e *Spywares* devem ser monitoradas. Isso permite que a análise se concentre nas atividades normalmente executadas por essas espécies, tornando o exame mais direcionado ao real comportamento da amostra. Ademais, na atualidade, os artefatos normalmente executam atividades de mais de uma família, o que torna a modelagem por meio do *FCM* mais fidedigna.

### 5.3. Tempo de Execução e Escalabilidade

Nos experimentos realizados, o algoritmo *K-Means* foi executado da seguinte forma: `kmeans(características, 9, nstart = 25)`, onde o parâmetro "características" representa os vetores obtidos do conjunto de amostras, o valor "9" corresponde ao número de *clusters* definidos para geração da saída e "nstart = 25" é o número de vezes diferentes que os *clusters* de partida são aleatoriamente gerados, a fim de que se possa escolher a melhor configuração para inicialização. Com relação ao *FCM*, este foi executado da seguinte maneira: `cmeans(características, 9, iter.max = 100, dist = "euclidean", m = 1.5)`, onde os dois primeiros parâmetros são idênticos aos do *K-Means*, "iter.max = 100" é o número máximo de iterações para agrupamento, "dist = euclidean" é a métrica para calcular a distância entre os exemplares (no caso, distância Euclidiana), e "m = 1.5" se refere ao índice de *fuzzificação*. A Tabela 8 mostra a medida dos tempos para a aplicação de cada algoritmo.

**Tabela 8. Tempo de execução medido para cada algoritmo.**

| Algoritmo | Tempo (s) |
|---|---|
| *K-Means* | 21.1 |
| *FCM* | 19.6 |
| Diferença | 2.5 |

Cabe ressaltar que, apesar do *K-Means* possuir complexidade inferior ao *FCM*, especificamente neste experimento, o parâmetro "nstart = 25", que possibilita uma aplicação mais robusta do *K-Means*, elevou consideravelmente seu tempo de execução.

Por fim, no caso de novas amostras a serem analisados, faz-se necessário reexecutar o método para agregar esses novos artefatos aos *clusters*. No entanto, todo o processo foi automatizado por meio de *scripts* em *Python* e R[8]

## 6. Conclusões e Trabalhos Futuros

Este artigo propõe a aplicação de um método baseado em lógica *fuzzy* para alcançar uma forma adequada de agrupar exemplares de *malware* modernos. Pôde-se observar a vantagem do uso do algoritmo *Fuzzy C-Means (FCM)*, cujas tabelas de pertinência permitem sugerir nomes aos grupo de exemplares de *clusters* e fornecem mais informações do que simples rótulos atribuídos por antivírus. Além disso, outro benefício de aplicação de lógica *fuzzy* em relação ao método de lógica *crisp* reside no fato de que os programas maliciosos não se limitam apenas a um comportamento específico de uma dada família,

[8]Disponíveis em: `https://drive.google.com/drive/folders/0B09rUNJplNMLTGVrb0VvZ0tvaDA?usp=sharing`.

 

ou seja, podem pertencer a várias delas ao mesmo tempo. Portanto, a lógica *fuzzy*, apresentada nos resultados deste trabalho, modela, de forma mais fidedigna, o real comportamento malicioso exibido durante uma infecção.

Como trabalhos futuros, pretende-se aprimorar o processo de rotulação de cada amostra e, consequentemente dos *clusters* produzidos. Nos experimentos, percebeu-se que o principal objetivo de um fornecedor de AV é detectar códigos maliciosos, sendo o processo de rotulação uma prioridade secundária. Logo, AVs não possuem um processo de rotulação confiável, tornando necessário o desenvolvimento de um novo método. De acordo os testes realizados neste trabalho, parece viável produzir rótulos baseados apenas nas características internas de cada amosta no *cluster*. Para isso, pode-se eleger as amostras mais representativas de cada *cluster* e estas devem ser analisadas a fim de se descobrir seu comportamento e posterior rotulação do *cluster*. Nessa abordagem, o rótulo de cada *cluster* não seria mais baseado em AVs. Com relação à análise de novas amostras, pretende-se utilizar um método supervisionado, uma vez que se pode comparar melhor agrupamentos conhecidos a priori. Planeja-se também executar uma quantidade maior de exemplares para se verificar a formação de outros *clusters*, os quais podem apresentar novos comportamentos. Almeja-se também melhorar o processo de definição da quantidade de *clusters* e distribuição das amostras nos agrupamento, de modo a se ter mais dinamicidade na geração de modelos de classificação.

## Referências

Andrade, C. A. B., de Mello, C. G., and Duarte, J. C. (2013). Malware automatic analysis. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 681 – 686. IEEE.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers Norwell, MA, USA, Utah State University, Logan, Utah, USA, 1 edition.

Bezdek, J. C., Ehrlich, R., and Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2–3):191–203.

Borges, V. R. P. (2010). Comparação entre as Técnicas de Agrupamento K-Means e Fuzzy C-Means para Segmentação de Imagens Coloridas. *XII Encontro Anual de Computação (EnAComp)*.

Damballa (2009). 3% to 5% of enterprise assets are compromised by bot-driven targeted attack malware. `http://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-61634867.html`. Acessado em junho de 2016.

Duarte, J. C., de Almeida Oliveira, F., dos Santos, J. C., and de Oliveira Neto, G. A. (2012). Framework de Aprendizado de Máquina (FAMA). `https://code.google.com/archive/p/fama/`. Acessado em junho de 2016.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience, New York, USA, 2 edition.

Dunn, J. C. (1974). A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3):32–57.

 

Firdausi, I., lim, C., Erwin, A., and Nugroho, A. S. (2010). Analysis of machine learning techniques used in behavior-based malware detection. In *Proceedings of the 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, ACT'10, pages Pages 201 – 203.

Huang, H.-D., Acampora, G., Loia, V., Lee, C.-S., Hagras, H., Wang, M.-H., Kao, H.-Y., and Chang, J.-G. (2013). Fuzzy markup language for malware behavioral analysis. In *On the Power of Fuzzy Markup Language*, pages 113–132. Springer.

Intel Security (2014). Relatório do mcafee labs sobre ameaças. `http://www.mcafee.com/br/resources/reports/rp-quarterly-threat-q3-2014.pdf`. Acessado em junho de 2016.

Iseclab (2015). Anubis - Analyzing Unknown Binaries. `http://analysis.iseclab.org/`. Acessado em junho de 2015.

MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations. *5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281 – 297.

Mangialardo, R. J. (2015). Integrando as análises estática e dinâmica na identificação de malwares utilizando aprendizado de máquina. Master's thesis, Mestrado em Sistemas e Computação, Instituto Militar de Engenharia, Rio de Janeiro.

Mohaisen, A. and Alrawi, O. (2013). Unveiling zeus: automated classification of malware samples. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 829–832. ACM.

Pirscoveanu, R.-S. (2015). Clustering Analysis of Malware Behavior. Master's thesis, Institute of Electronic Systems Department of Communication Technology at Aalborg University, Denmark.

Provataki, A. and Katos, V. (2013). Differential malware forensics. *Digital Investigation: The International Journal of Digital Forensics & Incident Response*, 10(4):Pages 311–322.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Salehi, Z., Ghiasi, M., and Sami, A. (2012). A Miner for Malware Detection Based on API Function Calls and Their Arguments. In Salehi, Z., editor, *Artificial Intelligence and Signal Processing (AISP)*, pages 563 – 568. IEEE.

Sami, A., Yadegari, B., Rahimi, H., Hamzeh, A., Hashemi, S., and Hamzeh, A. (2010). Malware detection based on mining api calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC'10, pages 1020 – 1025. ACM New York, NY, USA.

Swieskowski, P. and Kuzins, S. (2016). Ninite. `https://ninite.com/`. Acessado em fevereiro de 2016.

Thorndike, R. L. (1953). Who Belongs in the Family? *Pyschometrika*, 18(4):267–276.

VirusBulletin (2016). Vb100. `https://www.virusbulletin.com/testing/vb100/`. Acessado em fevereiro de 2016.

 