Metodologia de Detecção de *Malware* por Heurísticas Comportamentais

Neriberto Prado[1], **Ulisses Penteado**[1], **André Grégio**[2]

[1] Bluepex Security Solutions
Limeira – SP – Brasil

[2]Departamento de Informática (DInf)
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Malicious programs have evolved either in sophistication as in complexity level, increasing the rate of successful attacks against computer systems and their users. Such as any benign program, malicious ones need to interact with the underlying operating system so as to perform their intendend activities. Thus, there is a need for understanding which of the performed actions are related to infection procedures. These "suspicious" actions mold the malware's execution behavior, whose identification is decisive for detection. In this paper, we propose a methodology to detect malware based on behavioral heuristics. We also present tests and results of its application in actual samples.*

Resumo. *Programas maliciosos têm evoluído em sofisticação e complexidade, aumentando a incidência de ataques bem sucedidos contra sistemas computacionais e seus usuários. Como qualquer programa benigno, os programas maliciosos precisam interagir com o sistema operacional de forma a realizar as atividades pretendidas. Assim, faz-se necessário compreender quais das ações efetuadas estão envolvidas em processos de infecção. Tais ações "suspeitas" compõem o comportamento de execução do malware e sua identificação é crucial na detecção desses programas. Neste artigo, propõe-se uma metodologia para detecção de malware baseada em heurísticas comportamentais e apresenta-se os testes e resultados obtidos de sua aplicação em exemplares reais.*

1. Introdução

Programas maliciosos, também conhecidos como *malware*, são programas de computador que atuam de forma a subverter o sistema operacional ou a executar ações que beneficiem o atacante (por exemplo, roubo de credenciais da vítima). Dado que *malware*, em essência, é um tipo de programa como qualquer outro programa legítimo, é necessário conhecer que tipos de ações executadas por este podem estar envolvidas em processos de infecção. Tais ações, consideradas suspeitas, quando executadas de forma encadeada levam ao comprometimento da vítima, incorrendo em ataques contra a integridade, confidencialidade e/ou disponibilidade dos dados, do sistema ou do usuário-alvo em questão. A identificação de ações suspeitas e sua atribuição a um processo de infecção levam à detecção de uma execução, processo ou programa malicioso.

A detecção de programas maliciosos pode ser feita de forma estática ou dinâmica, isto é, por meio da análise do arquivo binário que representa o exemplar do *malware* ou da

execução de tal arquivo em um emulador (*engine*) ou ambiente controlado (*sandbox*). A análise estática é um procedimento que pode ser bastante custoso, devido ao uso de algoritmos de força-bruta (abordagem *greedy*) que tentam seguir todos os ramos em um fluxo de programa, exaurindo as possibilidades presentes em saltos e estruturas condicionais presentes nas instruções. De maneira mais simples, a análise estática pode prover informações importantes em uma pré-análise, seja através de dados extraídos do cabeçalho do *malware* (ofuscação, nomes de seções, entropia) ou por dados encontrados no conteúdo do binário (chamadas de funções efetuadas, porções de texto, endereços IP e de correio eletrônico, URLs, etc.). Entretanto, caso o *malware* sob análise esteja ofuscado por um empacotador ou cifrador, esta não terá sucesso a menos que o *packer* seja removido de antemão.

No caso da análise dinâmica, o exemplar de *malware* é efetivamente executado, ou seja, suas instruções são "acionadas" em um emulador por um número finito de ciclos ou em uma *sandbox* por alguns minutos, permitindo que a infecção ocorra. Dessa forma, o *malware* pode exibir seu comportamento e o mecanismo de monitoração pode obter as ações que são realmente feitas, evitando o custo de se tentar seguir todos os ramos presentes na análise estática e tornando possível a desofuscação do exemplar. A monitoração via análise dinâmica consiste em interceptar chamadas de sistema e de API em busca de ações suspeitas. Quando um certo conjunto de ações suspeitas é observado, é possível "marcar" o programa que originou essas ações e tomar alguma medida cabível, que varia entre emitir um alerta, colocá-lo em quarentena, desinfectar o sistema, entre outras. A observação das ações suspeitas depende da definição de heurísticas que representem comportamentos a serem correspondidos.

Neste artigo, introduz-se uma metodologia para detecção de programas maliciosos que se utiliza de heurísticas definidas com base em ações suspeitas, isto é, aquelas que são comumente executadas por *malware* ou têm o potencial de subverter os sistemas-alvo. No decorrer do trabalho, apresenta-se o modelo heurístico criado, as heurísticas propriamente ditas, um estudo de caso envolvendo a análise dinâmica de *malware* para extração dos comportamentos e os testes e resultados da detecção de programas maliciosos por meio da aplicação de aprendizado de máquina. Outras importantes contribuições deste trabalho incluem uma abordagem de detecção bastante simples em relação às disponíveis, bem como uma extensa revisão do estado da arte. O restante do artigo está organizado como segue: na Seção 2, são mostradas abordagens vistas na literatura para detecção de *malware* através de seu comportamento; na Seção 3, é apresentado o modelo heurístico proposto em detalhes, os comportamento definidos, suas descrições e riscos associados, bem como o algoritmo para detecção desses comportamentos em programas com base nas heurísticas propostas; na Seção 4, são mostrados os testes realizados para verificar a capacidade de detecção do modelo gerado e a discussão dos resultados obtidos; na Seção 5, são apresentadas as considerações finais deste trabalho.

2. Trabalhos Relacionados

[Christodorescu et al. 2005] descreveram um modelo de detecção de programas maliciosos baseado em *templates*. Estes são definidos para os seguintes comportamentos: *decryption-loop*—programas que se desofuscam/decifram em memória após execução—e *mass-mailing*—funcionalidade de envio de *spam* comumente associada a *worms*. A detecção por *templates* foi feita pela comparação dos grafos de controle de fluxo (CFG)

contendo as instruções obtidas do *malware* por *disassembling* via `IDA Pro`[1]. A abordagem almejou identificar principalmente *malware* polimórfico (capacidade de gerar mutantes), mas os testes foram limitados a três famílias de *malware* mutante da época e aos dois *templates* semânticos mencionados.

[Yin et al. 2007] apresentaram Panorama, um sistema para detecção automatizada de *malware* por meio de um processo de testes, monitoração e análise de programas em execução. A parte de testes foi feita pela inserção no sistema de dados sensíveis "marcados" não direcionados ao programa sob análise; a monitoração consistiu em identificar se o *malware* fez algum acesso a esses dados marcados e qual sua ação sobre eles (por exemplo, envio desse dado pela rede visando roubo de informações); a etapa de análise foi realizada sobre os grafos gerados do fluxo de dados marcados, aos quais foram aplicadas políticas definidas para detecção de comportamento malicioso. Essas políticas pretendiam detectar *malware* especificamente envolvido em vazamento de informações da vítima, incluindo senhas, texto, documentos, páginas Web e protocolos de rede.

[Ye et al. 2007] apresentaram o *Intelligent Malware Detection System* (IMDS), um sistema de classificação baseado em regras que age sobre as sequências de execução das APIs do Windows. O objetivo principal do IMDS foi o de detectar *malware* polimórfico ou desconhecido via associações encontradas em chamadas de API obtidas diretamente do binário em formato PE. Para tanto, calculou-se o suporte e a confiança relacionados ao encadeamento das APIs, atribuindo o rótulo "maligno" ou "benigno" ao exemplar que realizou tais chamadas. A interpretação do comportamento executado pelo conjunto de chamadas de API careceu de interpretação adicional. Os autores compararam os resultados obtidos na detecção via IMDS com alguns antivírus famosos e da técnica de classificação por associação *a priori* com outros algoritmos de aprendizado de máquina.

[Jacob et al. 2008] apresentaram a detecção comportamental de *malware* como um processo de inferência, uma vez que tentou-se identificar quais ações de um programa foram realizadas com intuito de atacar o sistema ou seus usuários. Os autores discutiram os problemas deste tipo de detecção que, embora permitisse a identificação de exemplares desconhecidos, requeria a definição de comportamentos capazes de separar adequadamente os programas benignos dos maliciosos. O processo de definição desses comportamentos foi complexo e envolveu a criação de diversos perfis, ou heurísticas, que detectassem tipos distintos de *malware*. Para abordar o problema, os autores fizeram um levantamento da literatura sobre o tema e criaram uma taxonomia voltada para a detecção de *malware* baseada em comportamento. A taxonomia proposta abrangeu a extração de dados via análise estática e dinâmica, atuação sobre instruções, funções, chamadas de sistemas, grafos de fluxo e abstrações algébricas, e detecção por algoritmos heurísticos, classificadores diversos, sistemas especialistas e verificação de modelos. Com isso, teve-se um panorama bastante completo do domínio entre 1994 e 2007.

[Griffin et al. 2009] apresentaram Hancock, um sistema para geração automática de assinaturas para detecção de *malware*. Hancock usava cadeias de caracteres de 48 bytes para identificar sequências de código maliciosas dentro de um arquivo, generalizando o processo de comparação por *hashes*, o qual em geral requer uma assinatura por exemplar. A fim de minimizar a quantidade de falsos-positivos, Hancock comparava as potenciais

[1] `https://www.hex-rays.com/products/ida/index.shtml`

assinaturas de 48 bytes com um modelo criado a partir de programas conhecidamente benignos, descartando aquelas que apresentam correspondência em tal modelo. Para gerar as assinaturas, o sistema dependia que o exemplar esteja desofuscado (a fim de que não fosse criada uma assinatura do *packer*) e que as instruções tivessem sido obtidas por *disassembling* (via `IDA Pro`). Testes do sistema resultaram em taxa de falsos-positivos próxima de zero, porém baixa cobertura de detecção, dada a ofuscação de exemplares e a variedade de famílias. Mostrou-se, com isso, que soluções para a geração automática de assinaturas, sem a intervenção humana e sem a execução do exemplar (análise dinâmica) ainda não são suficientemente maduras para lidar com o problema de detecção de *malware*

[Treadwell and Zhou 2009] propuseram uma abordagem para detecção por heurísticas cujo alvo é *malware* com *packers*. Esse tipo de exemplar, ao ser ofuscado, apresenta certos padrões em seu cabeçalho que podem prover pistas para sua identificação. Os autores se aproveitaram disto para criar detectores que atuavam antes do processo de desofuscação, impedindo assim a execução do programa malicioso. Com base em oito características extraídas estaticamente do cabeçalho PE de *malware*, foi calculado um nível de risco inicial baseado na presença dessas que, se alcançasse um determinado limiar, alertava o exemplar como suspeito.

[Bazrafshan et al. 2013] discutiram as limitações da detecção de *malware* por correspondência de padrões, isto é, assinaturas, bem como abordagens da literatura baseadas em heurísticas. Tais heurísticas podem ser baseadas em chamadas de funções de APIs, grafos de controle de fluxo, n-gramas, opcodes, ou apresentar características híbridas. Essas características foram utilizadas para se identificar o comportamento de programas maliciosos e benignos, aos quais são aplicados técnicas de aprendizado de máquina. Os autores expuseram as desvantagens das técnicas disponíveis na literatura, como grafos muito grandes, alto custo computacional, quantidade de regras criadas, conjuntos de treinamento pequenos, entre outras.

[Adkins et al. 2013] propuseram um algoritmo para detectar programas maliciosos por meio da descoberta de trechos de código reutilizados. Este algoritmo, denominado BBCP, opera em blocos básicos extraídos de binários, os quais são então processados de forma a gerar uma "assinatura". O objetivo foi o de se identificar variantes ou funções maliciosas compartilhadas entre executáveis. Os blocos básicos foram obtidos pelo `IDA Pro` e as instruções foram normalizadas e agrupadas em 4-gramas para a produção de *hashes*, os quais foram utilizados na comparação por similaridade (calculada pelo índice de Jaccard).

[Ahmadi et al. 2013] introduziram um método de detecção heurístico que se utilizava de mineração de padrões iterativos de chamadas de API. Os exemplares de *malware* foram executados em máquina virtual e as chamadas de API realizadas por estes foram monitoradas e armazenadas em traços comportamentais. Cada API foi mapeada para um identificador numérico único a ser utilizado em uma sequência de detecção. Cada sequência foi então representada por um conjunto de APIs chamadas em ordem e cuja frequência de ocorrência obedecia a um limiar determinado. A classificação de um programa como malicioso envolveu associar a ele uma ou mais sequências rotuladas como tal. Esse processo foi feito por meio do algoritmo *RandomForest*.

[Alazab 2015] se propôs a diferenciar programas maliciosos de benignos por meio

de assinaturas baseadas em chamadas de API extraídas através de *disassembling* pelo `IDA Pro`. Essa detecção foi feita calculando-se a distância entre sequências de chamadas de API e a frequência em que elas ocorreram em ambas as classes. Assim, foi possível obter a similaridade dos exemplares (maliciosos e benignos) avaliados, para os quais foi também aplicado o algoritmo KNN. Embora os testes, diferente de outros trabalhos, tenham sido feitos em Windows 7, a base de exemplares maliciosos era muito antiga para ser considerada relevante perante os exemplares atualmente observados.

[Prelipcean et al. 2015] apresentaram um método estatístico para detecção de comportamentos de execução específicos em programas. Para gerar os traços comportamentais, os autores executaram os exemplares (*malware* e programas "limpos") em ambiente controlado e obtêm suas chamadas de API. Conjuntos de chamadas presentes na sequência que compõe um traço, as quais podem levar a um comportamento malicioso, foram denominados "contextos". Verificou-se então a ocorrência dos contextos nos traços capturados e analisou-se a probabilidade de que tais contextos ocorressem em traços advindos de programas limpos. Dependendo da frequência de aparição dos contextos em exemplares desconhecidos e da probabilidade deles em traços limpos, tais exemplares puderam ser considerados benignos, suspeitos ou maliciosos. A taxa de detecção obtida com a aplicação do método foi próxima de 80%, com falsos-positivos menores que 2%.

[Khodamoradi et al. 2015] propuseram uma técnica para detecção de *malware* metamórfico baseada em assinaturas feitas a partir de estatísticas sobre contagem de instruções do programa analisado. As instruções foram obtidas por análise estática, sua frequência foi calculada e os opcodes foram utilizados como características para classificação por meio de seis algoritmos de árvores de decisão. O treinamento e a classificação feitos da maneira proposta foram rápidos, mas o método utilizado poderia ser facilmente subvertido por programas maliciosos evasivos que se utilizassem de substituição de instruções. Além disso, os exemplares utilizados na avaliação correspondem em sua maioria à *malware* antigo, os quais não são mais atuantes.

[Feng et al. 2015] apresentaram HRS, uma plataforma híbrida que fazia uso de modelos baseados em *hash*, regras e *Support Vector Machines* para detecção de *malware*. O primeiro modelo foi o tradicional utilizado, de comparar *hashes* com uma lista de exemplares conhecidamente maliciosos ou benignos, enquanto que os dois último pretendiam detectar exemplares desconhecidos por meio de heurísticas ou treinamento de classificadores gerados a partir de n-gramas advindos de *scanning* do binário. Os resultados do teste com as regras mostraram uma taxa de verdadeiros-positivos de 76.10%.

[Fan et al. 2016] introduziram uma plataforma para detecção de *malware* que se utilizava de mineração de dados para encontrar padrões em sequências de instruções extraídas de arquivos do tipo PE. O arquivo a ser analisado passava por um processo de *disassembling* seguido pelo mapeamento das instruções para um identificador numérico. Tais identificadores, na ordem em que aparecem, representavam um programa e foram filtrados de forma a se retirar instruções que não eram úteis para a detecção. De acordo com a frequência das instruções, calculou-se um peso para auxiliar na classificação entre maligno e benigno. Subsequências encontradas em *malware* foram utilizadas no processo de detecção, que se utilizou de uma variação do algoritmo "k-vizinhos mais próximos" (KNN) denominada por ANN, onde o parâmetro "k" foi escolhido automaticamente. Os resultados em comparação com outros classificadores baseados em distância foram mar-

ginalmente melhores.

Cabe ressaltar que quase a totalidade dos trabalhos relacionados é ou atrelado à análise estática com *IDA Pro*—dependente da qualidade do *disassembling* e prejudicados pela ofuscação presente em *malware* modernos—ou executado em ambientes virtualizados com o sistema operacional Windows XP, mesmo na literatura mais recente. O trabalho desenvolvido neste artigo se utiliza não só de heurísticas mais genéricas para definir comportamentos suspeitos, como executa exemplares mais atuais em sistema operacional Windows com kernel NT 6.x (Windows Vista até o atual 10)[2]. Por ser feito com base em análise dinâmica, evita-se que técnicas de ofuscação atrapalhem a extração de comportamento. Entretanto, como qualquer abordagem baseada na execução de *malware*, há limitações para o caso de exemplares que aplicam técnicas evasivas (anti-análise, anti-forense, `sleep`, etc.).

3. Modelo Heurístico

Uma heurística pode ser vista como um modelo, processo ou método composto por uma ou mais regras que visam encontrar a solução para um dado problema com base em informações altamente especializadas. O resultado da aplicação de um conjunto de heurísticas pode ser uma aproximação para um programa específico, por exemplo, quais são as ações executadas por um programa que determinam se este é malicioso ou não. O modelo heurístico proposto neste artigo visa agrupar características e informações que serão utilizadas para descrever uma ação suspeita genérica. Esse modelo servirá de base para a definição das ações suspeitas que, por sua vez, serão usadas como insumo para um algoritmo de análise comportamental de programas. O algoritmo recebe dados sobre a execução de um determinado programa monitorado (também chamado de "traço") e aplica regras de comparação a fim de identificar quais das heurísticas correspondem às ações observadas. Com isso, ao fim da execução do algoritmo de análise comportamental, obtém-se um vetor de comportamentos exibidos pelo programa monitorado. Tal vetor permite que se detecte se o programa é malicioso em um passo posterior.

O modelo heurístico desenvolvido trata-se de uma estrutura de dados e pode ser facilmente armazenado em um banco de dados para facilitar as atividades de gerenciamento (inserção, remoção e edição) destas. Para representar tal estrutura de dados, modela-se um conjunto de campos com tipos específicos (em forma de tabela) que possa especificar qualquer heurística definida, isto é, ações suspeitas genéricas. Assim, pode-se partir para o processo de detecção de comportamentos suspeitos exibidos por programas, visando identificar programas maliciosos ou programas que apresentam atividades que potencialmente danosas a um sistema "alvo" (máquina da vítima). A estrutura que especifica as características que uma dada heurística deve conter é ilustrada na Tabela 1.

3.1. Descrição das características

As descrições dadas a seguir detalham as características elicitadas (vistas na Tabela 1) para a definição geral de uma heurística de detecção comportamental.

- **Identificador da heurística**: número inteiro crescente que serve como identificador da heurística (e chave primária de uma possível tabela de heurísticas em um banco de dados);

[2]O estudo de caso aqui realizado utilizou-se do Windows 7.

Tabela 1. Modelagem dos campos (características) que representam uma heurística para identificação de comportamento suspeito.

Campo	Tipo
Identificador da heurística	número inteiro sequencial
Nome da heurística	rótulo composto separado por "_"
Descrição	explicação do comportamento exibido
Risco associado	número inteiro no intervalo [1..5]
Categoria	tipo geral do comportamento
Padrão para detecção	expressão regular

- **Nome da heurística**: o nome é composto pelo evento (ex.: modificação, roubo, evasão, etc.) e/ou comportamento exibido (ex.: persistência, exfiltração de credenciais, anti-debugging, etc.), alvo da ação do programa monitorado (ex.: Registro, sistema de arquivos, processos, etc.) e variação, representada por um número inteiro sequencial que indica variantes de comportamentos que podem ser enquadrados no mesmo nome. Esta característica foi inspirada na taxonomia de comportamentos suspeitos definida por [Grégio et al. 2015];
- **Descrição**: indica que tipo de ação suspeita o programa monitorado exibiu, permitindo que se tenha informações de mais alto nível sobre o comportamento apresentado e a extensão do dano causado por uma possível infecção;
- **Risco associado**: define o nível do risco relacionado à atividade monitorada, podendo servir para identificar uma ameaça ou compor um índice com o somatório dos riscos atribuídos a um determinado processo. Os valores de risco vão de 1 (menos grave) até 5 (mais crítico) e foram inspirados na tabela apresentada por [Grégio et al. 2016], na experiência dos autores em anos de análise de *malware* e na observação de que determinados comportamentos aparecem mais frequentemente associados a programas maliciosos;
- **Categoria**: define uma classe para explicitar o tipo de comportamento observado, permitindo o agrupamento de exemplares similares e a motivação por trás da referida ação;
- **Padrão para detecção**: consiste em uma string especialmente criada para servir como um meio rápido para se detectar a ocorrência de um comportamento suspeito. É a regra que aciona a heurística definida nos demais campos.

3.2. Detecção Comportamental

A detecção comportamental é feita pela definição de comportamentos suspeitos associados à execução de programas maliciosos, os quais são então mapeados cada qual para um padrão (expressão regular ou cadeia de caracteres representativa). Este, por sua vez, é usado para se criar um *template* heurístico cujo conjunto produzido é comparado com o traço comportamental de programas analisados dinamicamente. A ocorrência de um dado *template* ativa seu comportamento em um vetor associado ao programa em questão e, ao final dessa análise, tem-se todas as atividades suspeitas efetuadas por cada programa. Mais adiante, os vetores rotulados como associados a programas maliciosos ou benignos são utilizados no treinamento de um classificador. Os comportamentos definidos inicialmente, para os quais *templates* heurísticos foram gerados, são os seguintes:

1. **Evasão**: modificação da configuração de notificação das atualizações do sistema. (*Risco: 4*)
2. **Evasão**: modificação da configuração do *firewall* nativo do sistema. (*Risco: 4*)
3. **Evasão**: chama o comando `sc.exe` para modificar configurações ou aplicações do sistema. (*Risco: 3*)
4. **Evasão**: chama o comando `netsh.exe` para modificar configurações privilegiadas do sistema. (*Risco: 3*)
5. **Evasão**: chama um terminal para execução de comandos ou programas. (*Risco: 1*)
6. **Evasão**: remoção de evidências do comprometimento do sistema. (*Risco: 2*)
7. **Evasão**: desabilita a inicialização em modo de segurança. (*Risco: 3*)
8. **Subversão do sistema operacional**: altera chaves de Registro para modificar políticas de segurança vigentes. (*Risco: 5*)
9. **Subversão do sistema operacional**: altera chaves de Registro para instalar automaticamente novos controles ActiveX. (*Risco: 5*)
10. **Subversão do sistema operacional**: altera chaves de Registro para abrir automaticamente certos tipos de arquivo. (*Risco: 4*)
11. **Subversão do sistema operacional**: altera comandos ou opções nos menus de contexto de arquivos do Windows no Registro. (*Risco: 3*)
12. **Subversão do sistema operacional**: carrega um *driver* na vítima, que pode ser um *rootkit*. (*Risco: 5*)
13. **Subversão do sistema operacional**: tenta executar arquivo `.exe`, criando novo processo. (*Risco: 2*)
14. **Subversão do navegador**: altera configurações de zonas de segurança do Internet Explorer no Registro. (*Risco: 4*)
15. **Subversão do navegador**: insere barras de ferramentas ou *Browser Helper Objects* no Internet Explorer. (*Risco: 4*)
16. **Subversão do navegador**: altera configurações de servidor de proxy do Internet Explorer no Registro. (*Risco: 4*)
17. **Subversão do navegador**: altera o arquivo de *hosts* para evitar o acesso a *sites* legítimos ou redirecionar para *sites* maliciosos. (*Risco: 5*)
18. **Subversão do navegador**: carrega um arquivo do tipo PAC (*Proxy Auto Configuration*) para redirecionar a navegação do usuário, o que pode ser usado para roubar credenciais de Internet Banking. (*Risco: 4*)
19. **Subversão das configurações de Internet**: altera zonas de segurança do Internet Explorer no Registro. (*Risco: 4*)
20. **Persistência**: modifica o Registro ou sistema de arquivos para executar na inicialização do sistema. (*Risco: 3*)
21. **Persistência**: altera chaves de Registros do WinLogon para iniciar automaticamente com o Windows. (*Risco: 4*)
22. **Persistência**: altera arquivos .INI ou Registros equivalentes para iniciar automaticamente com o Windows. (*Risco: 4*)
23. **Manutenção**: verifica conectividade com a rede externa. (*Risco: 1*)
24. **Manutenção**: cria um objeto de sincronização do tipo `mutex`. (*Risco: 1*)
25. **Disfarce**: modifica arquivos legítimos para se disfarçar e executar ações suspeitas. (*Risco: 5*)

É importante notar que nenhuma dessas atividades sozinhas garante que o programa em execução é malicioso. Entretanto, quando analisadas sob a ótica da execução de um programa sabidamente malicioso ou quando consideradas em conjunto, pode-se inferir um maior grau de suspeição a tal programa e observá-lo mais atentamente.

Um exemplo de *template* criado seguindo o modelo proposto pode ser visto na Tabela 2. Essa heurística representa o comportamento da persistência, isto é, a capacidade de um programa executar após reinicialização ou desligamento da máquina. Programas maliciosos fazem uso desse comportamento para "sobreviverem" na vítima (em vez de residirem na memória somente quando executados da primeira vez). Pode-se observar que a ativação de uma dada heurística é passível de ocorrer por meio de mais de um padrão para detecção.

Tabela 2. Exemplo de aplicação do modelo heurístico na criação de um detector do comportamento de persistência via modificação de chaves pré-definidas do Registro.

Campo	Tipo
Identificador da heurística	004
Nome da heurística	persistencia_registro
Descrição	modifica o Registro para executar na inicialização do sistema
Risco associado	3
Categoria	persistência
Padrão para detecção	".*\Software\Microsoft\Windows NT\CurrentVersion\Run", ..., ".*\Software\Microsoft\Windows\CurrentVersion\RunOnce", ...

3.3. Algoritmo de Identificação de Comportamento Suspeito

Uma vez criados, os *templates* são utilizados para encontrar instâncias desses comportamentos de acordo com um algoritmo definido. O Algoritmo 1 ilustra os passos efetuados com a finalidade de se identificar comportamentos suspeitos em programas monitorados por análise dinâmica.

4. Testes e Resultados

Os testes foram realizados com 2.959 amostras divididas em 2.394 maliciosas e 565 benignas.rotuladas como maliciosas foram coletadas no decorrer de 2015 e são provenientes do laboratório de análise de *malware* da Bluepex. As amostras rotuladas como benignas incluem programas nativos do sistema operacional Windows, aplicações disponíveis para *download* em *sites* Web e programas do *SysInternals*[3]. Todas os programas foram submetidos para análise dinâmica em ambiente de testes implantado para validação dos *templates* heurísticos com base no sistema *Cuckoo Sandbox* [Guarnieri 2013]. A escolha desse sistema deve-se à facilidade de se customizá-lo para que, dadas as heurísticas implementadas, se verifique se tal comportamento é exibido pelo programa analisado. A configuração do *Cuckoo* foi feita sobre o emulador QEMU [Bellard 2005] com sistema

[3]https://technet.microsoft.com/en-us/sysinternals/bb545021.aspx

Algorithm 1: Identificação de comportamento suspeito

ENTRADA: Traço obtido da monitoração da execução de um programa ou chamadas de API capturadas em tempo real.
SAÍDA: Comportamento encontrado e pontuação de risco associada.
Seja $A = \{a_1, a_2, ..., a_N\}$ o conjunto de ações presentes em um traço de análise dinâmica ou interceptadas por um monitor de chamadas de API.
Seja $H = \{h_1, h_2, ..., h_M\}$ o conjunto de heurísticas definidas que representam ações suspeitas
risco $\leftarrow 0$
comportamentos $\leftarrow$ vetor cujas posições representam cada $h_i \in H$ e são inicializadas com 0
for $a_i \in A$ **do**
    for $h_i \in H$ **do**
        if $h_i \subset a_i$ **then**
            comportamentos$[h_i] \leftarrow 1$
            risco $\leftarrow$ *risco* $+h_i.risco$
return *comportamentos*,*risco*

operacional Windows 7. Cada exemplar permaneceu em execução por no máximo quatro minutos e gerou resultados que passaram por uma etapa de pós-processamento, a fim de se identificar os comportamentos exibidos. Os *templates* heurísticos aplicados aos traços de análise dinâmica produziram os resultados mostrados na Tabela 3.

É interessante notar que as heurísticas são ativadas tanto em *malware* como benignos. Isso já é esperado, pois um programa malicioso nada mais é do que um programa como outro qualquer, porém com intenção de efetivar um "ataque". O que muda é o conjunto de comportamentos exibidos e sua frequência para ambos os tipos de programas, o que fica bem explícito na tabela. As duas primeiras linhas da tabela, que possuem os valores mais altos para ambas as classes de programas, são importantes para a discussão da aplicação de heurísticas comportamentais na detecção de *malware*: nota-se que não se pode afirmar que a criação de um novo processo é uma atitude maliciosa, mas a maioria dos exemplares de *malware* o faz; os programas legítimos, por sua vez, não costumam criar novos processos diferentes dos seus próprios após execução inicial. Da mesma forma, enquanto que a criação de um objeto de sincronização (mutex) não é incomum, uma análise mais detalhada, por exemplo, obtendo-se seu nome, pode revelar uma determinada família de *malware*, pois estes geralmente fazem uso de *mutexes* para evitar reinfecções e *crashes*.

Sete dos 25 comportamentos definidos não foram identificados no conjunto de exemplares utilizado nos testes. Isso pode ser ocasionado por diversos fatores: nenhum programa realmente apresentou tal comportamento, o tempo de execução foi insuficiente, a monitoração é limitada, entre outros. Porém, a ausência de um dado comportamento não afeta a proposta deste trabalho, uma vez que as heurísticas podem ser implementadas em outros sistemas de análise dinâmica ou como um *engine* próprio. Para fins de validação desta proposta, é suficiente mostrar que há possibilidade de se diferenciar programas maliciosos de benignos por meio da detecção de seu comportamento. Assim, calculou-se

Tabela 3. Porcentagem de exemplares maliciosos e benignos que ativaram heurísticas comportamentais. O número entre parênteses nas linhas da coluna "Heurística" representa o comportamento apresentado na lista da Seção 3.2.

Heurística	Malware	Benigno
Subversão do SO (13)	87,3%	15,6%
Manutenção (24)	77,2%	17,5%
Subversão do navegador (16)	44,7%	1,4%
Evasão (7)	19,5%	1,2%
Persistência (20)	6,0%	1,6%
Subversão do SO (12)	4,6%	1,1%
Evasão (4)	4,5%	0,2%
Evasão (5)	2,3%	0,4%
Subversão do navegador (15)	1,6%	0,2%
Evasão (3)	1,6%	0,2%
Subversão do SO (10)	1,6%	0,2%
Manutenção (23)	0,8%	✗
Subversão do SO (8)	0,5%	0,2%
Persistência (22)	0,5%	✗
Evasão (6)	0,5%	✗
Subversão do navegador (17)	0,2%	✗
Subversão do SO (11)	0,1%	0,4%
Subversão do SO (9)	✗	0,2%

a média do valor de risco encontrado nos programas maliciosos e benignos, obtendo-se 5,77 para os primeiros e 0,75 para os últimos. Cabe ressaltar que os programas pertencentes ao *SysInternals* (parte do subconjunto dos benignos) causaram aumento na ativação das heurísticas para este grupo, pois executam atividades privilegiadas (como carregar *drivers*) ou fazem muitas alterações em Registros e no sistema de arquivos que podem ser consideradas "administrativas" (pela própria natureza dessas ferramentas), mas que também são comumente observadas em *malware*.

Conforme mencionado, a aplicação do algoritmo de detecção heurística a cada um dos traços de execução dos programa monitorado foi utilizada para se gerar um vetor binário de comportamentos, no qual os índices representam comportamentos e seus valores ("0" ou "1") a ausência ou presença, respectivamente, do comportamento em questão. A Listagem 1 ilustra um vetor de comportamentos associado a um programa malicioso e outro a um programa benigno[4].

Listagem 1. Exemplos de vetores comportamentais para exemplar malicioso (rótulo 'm') na linha 1 e benigno ('g') na linha 2.

```
1 0 ,0 ,0 ,1 ,0 ,0 ,1 ,0 ,0 ,0 ,1 ,0 ,0 ,1 ,0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,m
2 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,g
```

Os vetores rotulados servem de entrada para um classificador baseado no Teorema de Bayes, que por sua vez faz uso do conceito de probabilidade condicional: $P(A \mid B)$

[4]O arquivo CSV com os vetores utilizados neste artigo, bem como a lista de MD5 dos exemplares estão disponíveis em: `http://www.inf.ufpr.br/gregio/SBSeg2016/`.

é a probabilidade de que um evento "A" ocorra dado que o evento "B" já ocorreu. É possível derivar o Teorema de Bayes a partir de probabilidade condicional e probabilidade conjunta, chegando à seguinte fórmula [Downey 2012]:

$$P(A \mid B) = \frac{P(A)\,P(B \mid A)}{P(B)}$$

O classificador Bayesiano costuma unir rapidez na geração do modelo com bom nível de precisão, sendo capaz de atuar em casos nos quais há alta dimensionalidade [John and Langley 1995]. Para os testes realizados, usou-se a implementação do algoritmo *Naïve Bayes* disponível no *framework* de mineração de dados WEKA [Hall et al. 2009]. O parâmetro passado para treinamento e teste foi a validação cruzada utilizando 10 pastas, resultando em 88,44% de exemplares corretamente classificados. Consequentemente, 11,56% deles foram incorretamente classificados, em parte devido às já mencionadas ferramentas do *SysInternals*. A Tabela 4 mostra os resultados da classificação de maneira mais detalhada, considerando as duas classes de programas presentes no conjunto de dados.

Tabela 4. Resultados da classificação dos programas em *malware* ou benigno, onde TP corresponde aos verdadeiros-positivos e FP aos falsos-positivos.

TP	FP	Precisão	*Recall*	*F-Measure*	Área ROC	Classe
90,1%	18,4%	95,4%	90,1%	0,927	0,933	*Malware*
81,6%	9,9%	66,0%	81,6%	0,729	0,933	Benigno

A avaliação dos resultados da tabela mostram que, enquanto 90,1% dos programas maliciosos foram corretamente classificados como tal, 18,4% dos programas benignos foram considerados *malware*. Esse resultado pode ter um impacto grande no uso de um sistema de detecção automatizado por usuários. No entanto, deve-se considerar que os vetores usados para classificação foram gerados com base na execução do programa em uma *sandbox*, o que traz diversas implicações (como a taxa de interações com os programas benignos, bem diferente do uso legítimo feito por um ser humano) que podem não se tornar um problema na vida real. Mesmo assim, é um ponto importante a se levar em conta na implementação deste tipo de abordagem para detectores em tempo real, o que está fora do escopo do presente artigo. Se considerarmos exclusivamente as habilidades de detecção de programas maliciosos, a qualidade (calculada através da *F-Measure*) do classificador foi boa: 92,7%. Esse resultado também é corroborado pela área sob a curva ROC, onde quanto mais próximo de 1 for o valor, mais exatidão se tem.

5. Conclusão

Neste artigo apresentou-se o estado da arte em detecção de *malware* baseada em heurísticas comportamentais, discutindo-se as limitações das abordagens existentes na literatura. Foi proposto um modelo de detecção heurística dependente de comportamentos de alto nível, os quais descrevem ações feitas por um programa durante o processo de infecção da vítima. Foram definidos comportamentos comumente exibidos por *malware* em execução, aos quais foram atribuídos valores de risco. O cálculo do risco associado a programas maliciosos e legítimos mostrou que há uma diferença considerável entre ambos, podendo

servir no futuro para o estabelecimento de limiares dinâmicos que auxiliem na emissão de alertas quanto a execução de programas suspeitos. Introduziu-se também um algoritmo que faz uso de *templates* heurísticos a fim de identificar comportamentos pré-definidos durante a execução de programas. A avaliação de programas reais, rotulados como maliciosos ou benignos, serviu para a geração de vetores binários explicitando a presença ou ausência da observação de cada um dos comportamentos definidos. Tais vetores foram utilizados como entrada em um algoritmo de classificação Bayesiana, que obteve taxa de acertos da ordem de 88,4% no conjunto de dados disponível (exemplares coletados em 2015). Os resultados mostram que é viável aplicar heurísticas comportamentais de alto nível para a detecção de *malware* em tempo real, seja em um *engine* de monitoração *stand-alone* ou como complementação de antivírus por assinaturas. Trabalhos futuros incluem o desenvolvimento de um protótipo que atue em tempo real, de forma a se medir o impacto no desempenho—atenuado pelo fato de que, uma vez que o modelo está treinado, a classificação Bayesiana é praticamente instantânea e que os antivírus já realizam a monitoração das chamadas de sistema e/ou de APIs em nível privilegiado, bastando-se implementar algum tipo de dicionário de comportamentos que gere os vetores em memória—e a consideração de outros "níveis" de avaliação de comportamento, como nomes de arquivos, valores de chaves de Registro e objetos do tipo `mutex` pré-definidos, bem como o encadeamento de comportamentos. Vale ressaltar que a abordagem proposta, embora simples e efetiva, necessita ser trabalhada de forma a minimizar drasticamente a quantidade de falsos-positivos de *malware*, isto é, a detecção incorreta de programas benignos como maliciosos, a fim de que possa ser utilizada em modo *stand-alone* e sem a supervisão de uma terceira parte (usuário, mecanismo de segurança ou profissional).

Referências

Adkins, F., Jones, L., Carlisle, M., and Upchurch, J. (2013). Heuristic malware detection via basic block comparison. In *Malicious and Unwanted Software: "The Americas"(MALWARE), 2013 8th International Conference on*, pages 11–18.

Ahmadi, M., Sami, A., Rahimi, H., and Yadegari, B. (2013). Feature. *Computer Fraud & Security*, 2013(8):11–19.

Alazab, M. (2015). Profiling and classifying the behavior of malicious codes. *J. Syst. Softw.*, 100(C):91–102.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., and Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*, pages 113–120.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA. USENIX Association.

Christodorescu, M., Jha, S., Seshia, S. A., Song, D., and Bryant, R. E. (2005). Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, pages 32–46, Washington, DC, USA. IEEE Computer Society.

Downey, A. B. (2012). *Think Bayes: Bayesian Statistics Made Simple*. Green Tea Press.

Fan, Y., Ye, Y., and Chen, L. (2016). Malicious sequential pattern mining for automatic malware detection. *Expert Syst. Appl.*, 52(C):16–25.

Feng, Z., Xiong, S., Cao, D., Deng, X., Wang, X., Yang, Y., Zhou, X., Huang, Y., and Wu, G. (2015). Hrs: A hybrid framework for malware detection. In *Proceedings of the 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics*, IWSPA '15, pages 19–26, New York, NY, USA. ACM.

Grégio, A., Bonacin, R., de Marchi, A. C., Nabuco, O. F., and de Geus, P. L. (2016). An ontology of suspicious software behavior. *Applied Ontology*, 11(1):29–49.

Griffin, K., Schneider, S., Hu, X., and Chiueh, T.-C. (2009). Automatic generation of string signatures for malware detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 101–120, Berlin, Heidelberg. Springer-Verlag.

Grégio, A. R. A., Afonso, V. M., Filho, D. S. F., Geus, P. L. d., and Jino, M. (2015). Toward a taxonomy of malware behaviors. *The Computer Journal*, 58(10):2758–2777.

Guarnieri, C. (2013). Cuckoo sandbox. `http://www.cuckoosandbox.org/`. Acesso em junho/2016.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

Jacob, G., Debar, H., and Filiol, E. (2008). Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4(3):251–266.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *11^{th} Conference on Uncertainty in Artificial Intelligence*, pages 338–345.

Khodamoradi, P., Fazlali, M., Mardukhi, F., and Nosrati, M. (2015). Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In *Computer Architecture and Digital Systems (CADS), 2015 18th CSI International Symposium on*, pages 1–6.

Prelipcean, D. B., Popescu, A. S., and Gavrilut, D. T. (2015). Improving malware detection response time with behavior-based statistical analysis techniques. In *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 232–239.

Treadwell, S. and Zhou, M. (2009). A heuristic approach for detection of obfuscated malware. In *Proceedings of the 2009 IEEE International Conference on Intelligence and Security Informatics*, ISI'09, pages 291–299, Piscataway, NJ, USA. IEEE Press.

Ye, Y., Wang, D., Li, T., and Ye, D. (2007). Imds: Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 1043–1047, New York, NY, USA. ACM.

Yin, H., Song, D., Egele, M., Kruegel, C., and Kirda, E. (2007). Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 116–127, New York, NY, USA. ACM.

