SAUCY SPICE: uma nova abordagem eficiente para a detecção de malwares baseada em assinatura

Leonardo Gonçalves Chahud[2], João Pedro Favoretti[1], Rafael Rocha[1], Nilson Sangy Jr.[3], Filipe Verri[1], Idilio Drago[4], Lourenço Alves Pereira Jr.[1]

[1]Instituto Tecnológico de Aeronáutica — ITA

[2]Universidade de São Paulo — ICMC/USP

[3]Polícia Federal, Brasília — PF

[4]Università di Torino, Italia — UNITO

lchahud02@usp.br,{favoretti,rafaelror,verri,ljr}@ita.br

Abstract. *The development of security solutions capable of correctly identifying and blocking malware has never expressed itself with such need. Nevertheless, nowadays solutions carry significant limitating issues with special regard to performance and scalability. Therefore, in the present work we propose the SAUCY SPICE: a signature based security solution that identifies and blocks malware threats through a filtering policy that can be well generalized to several malware executables. It also has a detection module designed with high performance and efficiency in mind. Experimentally, the results have shown high accuracy on the identification of malwares and minimum overhead caused by the solution.*

Resumo. *O desenvolvimento de soluções de segurança capazes de identificar e bloquear ameaças de malware nunca foi tão necessário. Contudo, observam-se nas soluções atuais limitações, sobretudo de desempenho e escalabilidade, as quais prejudicam a resolução deste problema. Dessa forma, este trabalho apresenta o SAUCY SPICE: uma solução baseada em assinatura capaz de identificar e bloquear softwares maliciosos através da criação de políticas com alta capacidade de generalização, e um módulo de detecção focado em eficiência e desempenho. Experimentalmente, foi observado overhead mínimo e alta taxa de acerto na identificação e bloqueio dos malwares.*

1. Introdução

Ataques cibernéticos causados por infecções de *malwares* continuam sendo uma grande preocupação. Em 2022, A empresa de segurança Kaspersky divulgou um comunicado o qual mostrou que mais de 15% dos computadores conectados à *internet* em todo o mundo experimentaram ao menos um tipo de ataque relacionado à infecção por *malware* naquele ano [Kaspersky 2022]. Além disso, de todos os alertas de segurança da informação emitidos pelo Governo Federal entre janeiro de 2020 a junho de 2023, aproximadamente 40% foram direcionados a medidas de proteção contra *malwares*[1]. Para mais, identificou-se no Sistema de Criminalística da Polícia Federal que diversos órgãos da Administração

[1]https://www.gov.br/ctir/pt-br/assuntos/alertas-e-recomendacoes/alertas/colecao-alertas

Pública Federal (APF) foram vítimas de crimes causados por códigos maliciosos entre os anos de 2017 e 2021 [Federal 2021]. Desse modo, apesar dos avanços no quesito de segurança, os *malwares* persistem significativamente como um problema crônico, demandando o desenvolvimento de sistemas de detecção mais eficientes.

Como contraponto aos *malwares*, a maioria das ferramentas de proteção fazem uso de sistemas de detecção baseados em assinaturas, sendo estas definadas, para o escopo deste trabalho, como propriedades dos arquivos executáveis as quais podem ser utilizadas para a identificação dos *malwares* [Abusitta et al. 2021]. Entretanto, essa abordagem possui algumas limitações práticas, como trabalho intensivo de um ser-humano (o que impacta a escalabilidade de criação de regras) e a baixa capacidade de identificar amostras desconhecidas [Abusitta et al. 2021]. Além disso, a operação baseada em arquivo e o aumento exponencial de assinaturas com o passar do tempo, representam um desafio de eficiência para esse tipo de detecção [Botacin et al. 2022].

Como resposta a esses desafios, a detecção de *malwares* precisa evoluir no sentido de criar regras pequenas que representem o máximo possível de *malwares* e de se ter um mecanismo de geração automática de assinaturas [Aslan and Samet 2020]. Nesse sentido, trabalhos anteriores obtiveram êxito em criar mecanismos de geração automática de regras para *malwares* polimórficos/metamórficos [Campion et al. 2021, Razeghi Borojerdi and Abadi 2013, Tang et al. 2009]. Outros trabalhos também tiveram sucesso em criar mecanismos de criação automática de assinaturas em cenários específicos. Para além do enfoque na criação de regras, trabalhos anteriores buscaram otimizar, via *hardware*, o processo de detecção [Botacin et al. 2022].

Buscando resolver as lacunas existentes, apresentamos o SAUCY SPICE (weightlesS, AUtomatiC and Yara-based Security Policy Implementation and Compliance Enforcement), uma solução que visa combater três problemas da detecção baseada em assinaturas: baixa eficiência, quantidade excessiva de regras e necessidade de analista humano para sua criação. Para tal, o SAUCY SPICE dispõe de dois módulos. O primeiro deles, o módulo de criação de assinaturas, é capaz de gerar automaticamente um pequeno conjunto de assinaturas que representam grupos de *malwares*). O outro módulo, responsável pela detecção, funciona totalmente em *kernel mode*, de modo a reduzir *overhead* computacional.

A solução, além de avaliada em *dataset* público, foi também testada em um cenário de ”mundo real”, em que o SAUCY SPICE foi testado na detecção de *malwares* já utilizados contra a APF. Como principais contribuições deste trabalho, podem ser citados: a) uma nova abordagem para criação automática de regras genéricas (utilizando *features* genéricas e funcionando para qualquer classe ou família de *malware*); b) uma nova abordagem de funcionamento para módulo de detecção baseado em assinaturas, funcionando totalmente em *kernel mode*; c) avaliação do desempenho da solução frente aos *malwares* que já foram utilizados em crimes contra a APF.

2. Trabalhos relacionados

Diversos trabalhos anteriores propuseram variadas formas de se gerar regras gerais que, a partir de um conjunto diminuto de amostas, representassem todas as suas variações metamórficas ou polimórficas, isto é, aquelas que se automodificam, com base em criptografia ou não, a cada replicação, na tentativa de dificultar a identificação por soluções de

segurança.[Campion et al. 2021, Razeghi Borojerdi and Abadi 2013, Tang et al. 2009]. Diferentemente do objetivo do SAUCY SPICE, as regras criadas por tais trabalhos limitam-se a generalizar as variações polimórficas ou metamórficas.

[David and Netanyahu 2015] criaram uma nova abordagem para gerar automaticamente assinaturas de *malwares* utilizando Deep Belief Network (DBN). No entanto, por utilizar características dinâmicas, a técnica depende da execução das amostras maliciosas em ambiente de *sandbox* e, além disso, técnicas de *deeplearning* sofrem conhecidamente de baixa explicabilidade, o que pode prejudicar o entendimento das regras geradas. Por outro lado, [Feng et al. 2017] propuseram uma solução para gerar automaticamente regras por meio da identificação de Suspicious Common Subgraph (MSCS) que consigam representar os malwares de uma mesma família. Entretanto, a solução proposta é específica para sistemas Android, pois utiliza como base o Inter-component Call Graph (ICCG).

Por fim, buscando melhorar o desempenho de detecção de *malwares* por assinaturas, [Botacin et al. 2022] propuseram uma solução assistida por *hardware*. A abordagem lança a mão de assinaturas em *hardware* para realizar uma triagem e utiliza assinaturas em *software* somente quando a amostra é suspeita. Diferentemente desse tipo de solução, a otimização de detecção do SAUCY SPICE é totalmente realizada em *software*.

Como observado, o presente trabalho diferencia-se dos demais por apresentar uma abordagem não supervisionada capaz de derivar uma visão filogênica das amostras a partir de análise estática. Com isso, é possível obter uma política de segurança implementada a partir de um mecanismo de controle de admissão que executa integralmente em *kernel-mode* no MS Windows. Mais ainda, foi possível testar a efetividade do SAUCY SPICE nos *malwares* que afetaram a administração pública federal do Brasil.

3. SAUCY SPICE

A Figura 1 ilustra a arquitetura do SAUCY SPICE, composta de dois módulos: Criação de Politica de Segurança da Informação (ou módulo de criação de assinaturas) e Aplicação de política de Segurança da Informação (ou módulo de detecção).

Para a criação de políticas, primeiramente é necessário que sejam extraídas *features* (características) que representem amostras de binários conhecidos. Em seguida, esse conjunto de *features* é carregado no DAMICORE, principal componente do módulo de criação de assinaturas e responsável por, de maneira não supervisionada, agrupar amostras levando em consideração a semelhança de suas *features*. Por fim, o componente de criação de regras leva em consideração um parâmetro passado pelo usuário (ϵ) para criar as assinaturas em um formato Yara[2] que representem os *clusters* de amostras semelhantes. Desse modo, é possível gerar, de maneira automática, um conjunto diminuto de regras capazes de representar um grande conjunto de amostras.

Já a aplicação da política de segurança gerada se dá por meio de um *driver* que intercepta a execução de binários no sistema operacional (SO). Ao interceptar a execução, o *driver* verifica se as *features* do arquivo coincidem com as regras geradas e, a depender do resultado, bloqueia ou permite a execução do arquivo. Com o intuito de ser mais eficiente, todo o funcionamento do módulo de detecção funciona no modo *kernel*.

[2]`https://yara.readthedocs.io/en/stable/index.html`

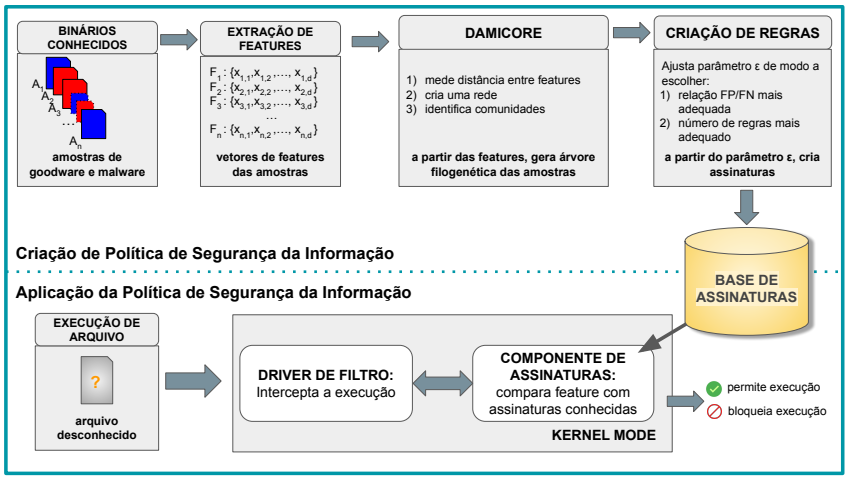


Figura 1. Diagrama do funcionamento do SAUCY SPICE.

A seguir serão aprofundados alguns conceitos necessários para compreender melhor o funcionamento da solução proposta.

3.1. Módulo de criação de políticas

O funcionamento do módulo de criação de políticas depende basicamente do agrupamento das amostras em *clusters* e da conseguinte criação de regras que identificam tais *clusters*.

Nas próximas subseções, será detalhado o funcionamento dos principais componentes do módulo de criação de assinaturas (componentes DAMICORE e criação de regras), e do módulo de detecção.

3.1.1. Aglomeração de Amostras

A solução utiliza o algoritmo de clusterização não supervisionado DAMICORE [Sanches et al. 2011] para encontrar amostras semelhantes antes de identificar a assinatura de cada *cluster*. A escolha do DAMICORE deve-se ao fato de ele já ter apresentado bons resultados em outras tarefas, como a de agrupamento de códigos semelhantes [Pinto et al. 2017], e de ser um *framework* genérico e flexível o suficiente para ser aplicado no agrupamento de amostra maliciosas.

O design original proposto para o DAMICORE possui três fases no fluxo de execução: (1) Cálculo da matriz de distâncias, (2) Criação de uma rede com as amostras e (3) Identificação de comunidades. No cálculo da matriz de distâncias, é utilizado um método chamado de Normalized Compression Distance (NCD), enquanto na criação da rede com as amostras é utilizado um algoritmo chamado de Neighbor Joining (NJ) e no último passo é utilizado um algoritmo chamado de Fastgreedy.

Cálculo de matriz de distâncias. Essa etapa é utilizada para criar uma matriz com as distâncias dois-a-dois das amostras. O método NCD é utilizado para realizar o cálculo da distância normalizada, retornando o valor 1 caso não haja nenhuma semelhança e 0 caso haja total semelhança. Esse algoritmo utiliza qualquer função de compressão Z(x), como

gzip e ppmd, que recebe um arquivo x como entrada para avaliar a quantidade de espaço de memória que é salvo ao compactar as duas amostras. Ao considerar que o valor de Z(x) é dado pelo tamanho do arquivo após a compressão, e xy representa a concatenação de dois arquivos x e y, a distância pode ser calculada a partir da seguinte equação:

$$NCD(x, y) = \frac{Z(xy) - min\{Z(x), Z(y)\}}{max\{Z(x), Z(y)\}}$$

Criação de uma rede de amostras. Ao utilizar a informação sobre a matriz de distâncias no agrupamento das amostras, é necessário utilizar uma abordagem diferente em relação à comum prática de representar cada amostra com coordenadas cartesianas e utilizar algoritmos como K-Means ou DBScan. A única forma de utilizar uma matriz de distâncias para entender a relação entre as amostras é a partir de um grafo e, com uma grande quantidade de amostras, esse grafo é chamado de rede complexa. Além disso, com a utilização do NCD para calcular as distâncias entre arquivos, não é comum encontrar arquivos completamente distintos e, portanto, com distância zero. Dessa forma foi utilizado o algoritmo Neighbor Joining (NJ) [Saitou and Nei 1987] para criar uma rede a partir da matriz distâncias calculada. Esse método verifica, a cada iteração, pares de vértices para encontrar qual par deve ser conectado no grafo e possui como resultado uma árvore filogenética que demonstra a similaridade entre as amostras. Uma árvore filogenética ou filogênica é utilizada neste contexto de análise de *malware* para explicitar o estudo das relações de diferentes de famílias de malware e, através de um grafo, representar uma árvore binária. Os nós que bissectam são nós auxiliares enquanto as amostras são representadas pelas folhas da árvore.

Identificação de comunidades. A medida de aglomeração no escopo de redes complexas é chamada de modularidade e pode ser calculada utilizando a topologia da rede e as classes de cada vértice da rede. O método Fastgreedy[Newman 2004] utiliza o paradigma guloso de programação, que foca em encontrar uma solução ótima local ao invés de encontrar a solução ótima global, o que possibilita encontrar resultados rápidos em grandes redes. O primeiro passo do algoritmo é atribuir classes diferentes para cada nó da rede e, a cada iteração, nós vizinhos são unidos em uma mesma classe caso a modularidade da rede toda aumente. Seguindo essa ideia, o algoritmo interrompe a execução caso não haja nenhuma mudança possível que aumente a modularidade da rede.

O DAMICORE é versátil o suficiente para receber qualquer tipo de entrada, podendo até mesmo realizar o agrupamento a partir da própria sequência de bytes do executável. Por esse motivo, o SAUCY SPICE pode utilizar quaisquer dos tipos de *features* normalmente utilizados em detecção de *malware*, tais quais sequências de bytes, strings ou chamadas API. Além disso, da forma como foi concebido, o SAUCY SPICE utiliza tanto amostras benignas quanto maliciosas para realizar os agrupamentos.

3.1.2. Criação de Regras

Na seção seguinte será destacado: como a informação sobre os agrupamentos foi utilizada para identificar assinaturas para algumas das classes do agrupamento; como essas

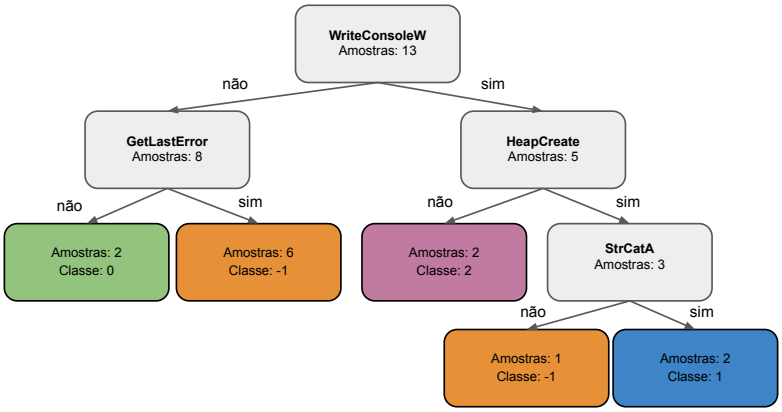


Figura 2. Árvore de decisão criada para diferenciar quatro classes $C = \{-1, 0, 1, 2\}$ baseado nas funções da IAT das amostras. As amostras da classe 0 (Bloco Verde) não possuem as funções `WriteConsoleW` e nem `GetLastError`.

Tabela 1. Assinaturas de cada classe pelo método da árvore de decisão.

Classe	Regra
0	$\neg$`WriteConsoleW` $\wedge$ $\neg$`GetLastError`
1	`WriteConsoleW` $\wedge$ `HeapCreate` $\wedge$ `StrCatA`
2	`WriteConsoleW` $\wedge$ $\neg$`HeapCreate`
-1	($\neg$`WriteConsoleW` $\wedge$ `GetLastError`) $\vee$(`WriteConsoleW` $\wedge$ `HeapCreate` $\wedge$ $\neg$`StrCatA`)

assinaturas se traduzem para as regras; e qual parâmetro foi utilizado para selecionar quais classes devem ser identificadas por regras.

Identificação de assinaturas Conforme visto, o DAMICORE cria os agrupamentos identificando um conjunto de *features* (e seus respectivos valores) que são relevantes para identificar cada grupo. Em outras palavras, a saída do DAMICORE pode ser entendida como grupos (ou classes) e as respectivas *features* que os caracterizam. Tal estrutura pode ser entendida como uma árvore de decisão, em que as folhas representam as classes identificadas pelo DAMICORE e cada um dos nós de decisão representam algum tipo de verificação sobre as *features* identificadas como relevantes para a aglomeração. A Figura 2 ilustra um exemplo dessa árvore de decisão separando treze amostras em quatro classes a partir da presença ou não de determinada característica.

Dessa forma, é possível utilizar as sequências de condições para descrever cada um dos nós folhas que representam uma determinada classe. Com isso, o conjunto de condições de *features* que identificam determinada classe pode ser entendido como a sua assinatura. Seguindo o exemplo dado, cada uma das classes utilizadas na árvore da Figura 2 podem ser representadas logicamente na Tabela 1. Essas regras podem ser unidas em um arquivo de descrição de assinaturas, tal como o formato YARA, e este pode ser utilizado para identificar binários que tenham *features* associadas com a classe identificada.

Como o *dataset* utilizado pelo SAUCY SPICE contém tanto amostras benignas

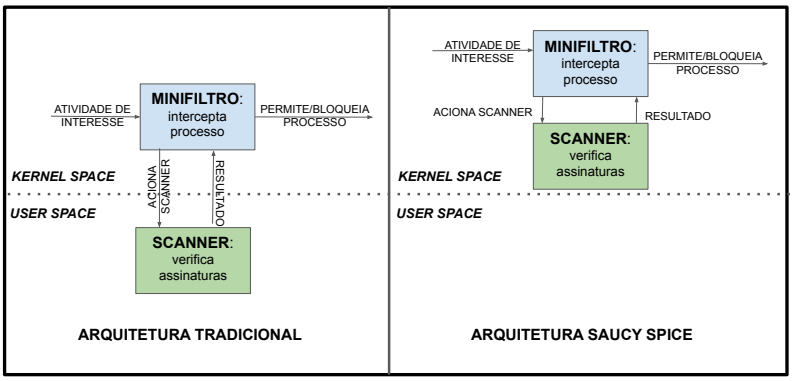


Figura 3. Diferença entre a arquitetura de detecção padrão e a do SAUCY SPICE.

quanto maliciosas, entre os grupos que são identificados, existem classes que representam amostras benignas. Apesar disso, apenas regras que caracterizem amostras maliciosas são geradas pela solução. No caso do exemplo dado na Figura 2, a classe identificada como -1 representaria amostras consideradas como benignas, enquanto as classes 0, 1 e 2 representariam amostras maliciosas. Ou seja, neste caso, apenas regras para as classes 0, 1 e 2 são geradas. A seguir será explicado como as assinaturas maliciosas são selecionadas.

Seleção de Assinaturas Conforme mencionado, as classes identificadas pelo SAUCY SPICE podem conter somente amostras maliciosas, somente amostras benignas ou uma combinação de ambas. Em outras palavras, é possível dizer que todo agrupamento possui as razões r_B de amostras benignas e r_M de amostras malignas, de forma que $r_B + r_M = 1$. Para selecionar classes que sejam representativas de amostras maliciosas, foi definido o parâmetro $\epsilon \in [0, 1]$, que representa a proporção de amostras maliciosas em uma classe. Dessa forma, toda classe que tiver $r_M > \epsilon$ será selecionada para gerar assinaturas.

3.2. Módulo de Detecção

Um *driver*, por definição, é um componente de *software* que realiza a comunicação entre um periférico (hardware) e o sistema operacional. O módulo de detecção desenvolvido consiste em um *software driver* do tipo Windows Driver Model (WDM), que funciona integralmente em *kernel space*. A solução foi desenvolvida para o Windows tendo em vista a maior relevância e utilização deste sistema. Esta decisão foi apoiada também sob a perspectiva de que os ataques direcionados à APF tiveram como alvo sistemas Windows.

O SAUCY SPICE utiliza o *driver* para acessar, manipular e mudar um curso de ações privilegiadas no sistema. Dessa forma, é possível interromper a execução/criação dos processos no sistema para que seja realizada a comparação do arquivo com a base de assinaturas de *malwares*. Cabe ressaltar que a verificação das assinaturas também é executada pelo *driver*, no espaço do *kernel*.

Tal modo de atuação difere-se do que ocorre atualmente nas soluções tradicionais de antivírus em tempo real. Tais soluções funcionam de maneira híbrida: um minifiltro intercepta atividades de arquivos no *kernel space* que, em seguida, aciona o verificador de assinaturas (scanner), que funciona em modo usuário[Mohanta and Saldanha 2020]. A Figura 3 ilustra a diferença entre as duas abordagens.

A decisão de implantar o *driver* somente em *kernel space* se deu por motivo de eficiência, buscando-se alta performance e mínimo *overhead* computacional. Com esta implementação, não há a troca de contextos entre o modo *kernel* e o modo usuário, minimizando assim, o impacto do *driver* sobre a máquina [Minghao et al. 2007].

Cabe ressaltar que o escopo do módulo de detecção desenvolvido é focado somente em arquivos executáveis e, portanto, não há a análise de arquivos de *script* como *visual basic*, *powershell*, *javascript*, etc.

4. Metodologia

Nessa Seção será explicado como o SAUCY SPICE foi implementado e como foram concebidos os experimentos para avaliar o desempenho da solução.

4.1. Implementação do SAUCY SPICE

4.1.1. *Feature* utilizada

Embora a solução seja agnóstica em relação ao tipo de *feature* utilizada para caracterizar os binários, optou-se por fazer a implementação do SAUCY SPICE utilizando somente chamadas API como *feature*. Conforme pode ser observado em trabalhos anteriores, esse tipo de *feature* é capaz de inferir comportamentos de binários e já foi largamente utilizada com sucesso em diversos trabalhos anteriores de detecção e de classificação de *malware*[Abusitta et al. 2021, Singh and Singh 2021].

Nesse sentido, a extração das *features* foi realizada de maneira estática, a partir da leitura da Import Address Table (IAT). Os binários passaram a ser representados por um vetor ordenado, cujos índices representam a existência ou ausência de determinada chamada API (chamada indistintamente de *import* neste trabalho) em sua IAT.

Para determinar esse vetor ordenado, foi necessário primeiramente identificar todas as chmadas API presentes nas IAT das amostras do *dataset* de treino, que compõem o vocabulário V de *imports*. Sendo n o número de amostras e I_i o conjunto de *imports* da amostra i, é possível representar o conjunto V da seguinte forma:

$$V = I_1 \cup I_2 \cup I_3 \cup \cdots \cup I_n \tag{1}$$

A partir de V, é definida uma ordem fixa para os *imports*, de modo que cada chamada API esteja associada a um índice j de V. Utilizando esse ordenamento fixo, cada *import* V_j passa a ser mapeado, de maneira bijetiva, por um índice j de F. Com isso, para representar uma amostra i, é criado um vetor F_i de tamanho $|V|$, em que cada índice j de F_i representa a presença ou ausência de determinado *import* da seguinte forma: $F_i^j = 1$, se a chamada API V_j está presente na tabela IAT de i; ou $F_i^j = 1$, no caso contrário.

Feita essa operação de extração de *features* para as n amostras do *dataset*, os n vetores resultantes são exportados para n arquivos de saída, cada um contendo uma sequência de $|V|$ bytes 0 e 1 representando os *imports* daquele arquivo. Esse conjunto de arquivos é fornecido como entrada para o DAMICORE.

4.1.2. DAMICORE

Para realizar a implementação do DAMICORE, utilizou-se como base um código aberto disponível na *internet*[3]. Após alguns testes iniciais, foi verificado que o método original do DAMICORE para cálculo de matriz de distâncias (NCD) estava prejudicando a identificação dos agrupamentos. Optou-se então por modificar o código e substituir o NCD por outra técnica de melhor desempenho. Após testes utilizando distância de Hamming e similaridade de cosseno, esta última técnica foi a que apresentou melhores resultados, gerando agrupamentos mais homogêneos, e foi escolhida para implementação final.

4.1.3. Implementação do módulo do *driver*

O *driver* foi implementado para, ao ser inicializado, ler o arquivo de regras gerado na etapa de criação de políticas e copiá-lo para a memória. Após esse processo, o *driver* utiliza a API do Windows (*PsSetCreateProcessNotifyRoutineEx*) para realizar o *hook* de processos e o fluxo de criação de novos processos no sistema operacional. Ao interromper a execução do processo, o *driver* identifica o caminho em disco do respectivo executável e extrai as *features* do binário. Finalmente, a partir das informações sobre a existência ou não de chamadas API na IAT do binário, o *driver* insere os valores *true/false* em cada assinaturas existente e, resolvendo um problema de lógica booleana, identifica se o binário coincide com alguma regra. A figura 4 ilustra o *pipeline* de funcionamento do *driver*.

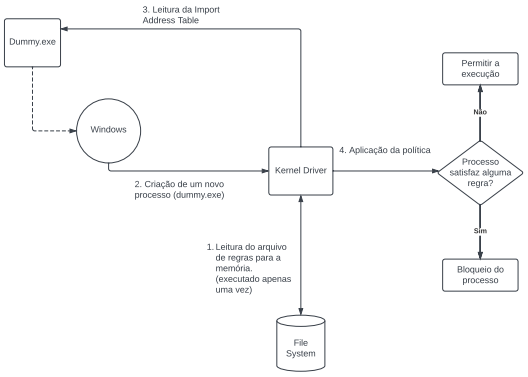


Figura 4. *Pipeline* de funcionamento do *driver* com o sistema operacional

Como discutido anteriormente, o desenvolvimento do módulo de detecção se deu somente em *kernel mode* para diminuir o overhead da solução sobre o sistema. Cabe ressaltar que a identificação e implementação de outras posíveis formas de melhorar o desempenho do proceso de detecção (como algoritmos e estruturas de dados mais eficientes), são questões ortogonais à proposta deste trabalho.

4.2. Experimentos

Nesta seção serão discutidos pontos avaliados para tomadas de decisão a respeito do método de avaliação de desempenho do SAUCY SPICE, assim como alterações reali-

[3] `https://github.com/brunokim/damicore-python`

zadas durante o pipeline de execução do DAMICORE.

4.2.1. Obtenção do(s) Dataset(s)

Como fonte de amostras maliciosas para os experimentos de validação da solução, foi utilizado o MalwareBazaar[4], que é uma coleção aberta de amostras de *malwares* rotuladas por especialistas e profissionais da área. Foram baixadas as amostras de 2021 durante os meses de Janeiro e Fevereiro, compreendendo 3.000 amostras de *malware*, todas filtradas para obter somente amostras não estaticamente compiladas. Além disso, para amostras benignas foram utilizados os binários presentes na imagem do Windows Server 2022, das quais 3.000 foram utilizadas para efetuar os testes de desempenho de detecção do modelo.

Além do *dataset* supracitado, foi criado um outro conjunto com 48 amostas maliciosas utilizadas em crimes cibernéticos contra a APF entre os anos de 2017 e 2021. Este último *dataset* foi utilizado para avaliar, em um cenário "mundo real", a capacidade do SAUCY SPICE em melhorar a segurança cibernética da Admnistração Pública Federal.

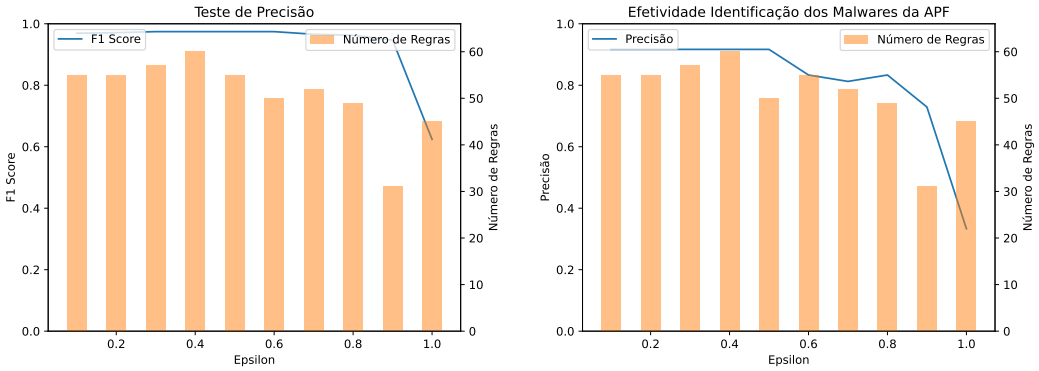
4.2.2. Validação da solução

Qualidade das regras geradas Uma das relevâncias do modelo proposto pelo SAUCY SPICE é a habilidade de criar poucas regras para descrever um grande número de amostras de treinamento. Esse resultado é relevante para generalizar as assinaturas e identificar *malwares* que não estão relacionados com a amostra de treinamento. Em vez de utilizar assinaturas específicas de cada amostra de treinamento, o modelo permite generalizar essas assinaturas para grupos de amostras maliciosas, que poderiam ser criadas manualmente por analistas de *malware*, porém com muito mais esforço.

Para mensurar qual a efetividade de utilização dessas assinaturas em amostras fora do conjunto de treinamento, foi utilizado todo o *dataset* contendo 6.000 amostras, sendo que foi realizada uma divisão de 90% das amostras utilizadas para treinamento e 10% das amostras utilizadas para testes. Como medida foi utilizado o F1-Score de modo a sumarizar os resultados de precisões obtidos para amostras de teste benignas e malignas. Ao mesmo tempo foi obtido o número de regras geradas para cada valor de ϵ para entender a significância da simplificação das regras geradas em comparação com conjunto de assinaturas para cada amostra maliciosa.

Eficiência do *driver* De modo a analisar o impacto da performance ocasionado pelo módulo de detecção (*software driver*), foram realizados *benchmarks* em um máquina virtual (Vmware) com o Windows 10 Pro instalado. Os *benchmarks* utilizados contemplam as seguintes soluções: Geekbench6, Novabench, 7Zip, Cinebench e Aida64. Todos foram executados com as opções que enfocam o *benchmark* da CPU visto que este seria, em teoria, o aspecto mais requisitado do sistema para a execução do *driver*. No caso de *benchmarks* os quais possuem como saída diversas métricas em relação a CPU (MIPS, pontuação, etc), como é o caso do Aida64, foi feita a média aritmética dessas medidas. Ademais, cada *benchmark* foi executado 10 (dez) vezes sem a presença do *driver*, e outras

[4]https://bazaar.abuse.ch/



(a) MalwareBazaar. (b) Ransomware da APF.

Figura 5. Resultados obtidos de malware oriundos de fontes distintas.

10 (dez) vezes com o *driver* carregado no sistema com o objetivo de minimizar a variação padrão encontrada entre as execuções dos *benchmarks*. Além disso, o sistema foi reinicializado a cada execução de *benchmark* para evitar a influência da organização de memória na execução dos próximos *benchmarks*. Assim, foi possível ter uma medição precisa e mais condizente com a realidade, permitindo aferir o impacto do *driver* no sistema.

4.2.3. Avaliação da solução em cenário real

Com o intuito de verificar o potencial da solução em melhorar a proteção da APF, foi realizado um último teste para medir quantos dos *malwares* utilizados nos crimes apurados pela PF entre 2017 e 2021 poderiam ser bloqueados pelo SAUCY SPICE. Para isso, garantiu-se, por meio de verificação de *hash*, que nenhum dos *malwares* utilizados pelos criminosos estava no *dataset* gerado a partir do MalwareBazaar. Em seguida, foi executado o módulo de criação de política do SAUCY SPICE e verificada, para diversos valores de ϵ, a quantidade de amostras utilizadas pelos criminosos que poderiam ser identificadas.

5. Resultados e discussões

5.1. Qualidade das regras criadas

A Figura 5(a) ilustra o resultado do F1 Score, métrica definida como a média harmônica entre precisão e revocação, e o respectivo número de assinaturas geradas. Conforme pode ser observado, a solução produziu um resultado de $F1\,Score \geq 0.9$ no intervalo $\epsilon \leq 0.9$ com valor máximo $F1\,Score = 0.98$ para $\epsilon = 0.1$. Além disso, pode-se observar que o número de assinaturas geradas ficou entre 30 e 60, não tendo esse valor uma relação direta com o parâmetro ϵ. Esse resultado enfatiza a eficácia do modelo SAUCY SPICE para reduzir o número de assinaturas obtidas para um número grande de amostras, sendo que nesse experimento foi possível caracterizar um grupo de 3000 amostras malignas com menos de 60 amostras.

5.2. Eficiência do driver de bloqueio

A Tabela 2 resume a pontuação média da performance obtida após as dez execuções em cada um dos benchmarks. Quanto maior a pontuação obtida, melhor o desempenho durante o experimento, ou seja, menos uso de CPU.

Conforme pode ser observado, houve impacto insignificante no desempenho da CPU quando o *driver* foi ativado. No *benchmark* Novabench, os desempenhos com e sem utilização do *driver* foram idênticos. Já nos *benchmarks* Aida64 e Geekbench 6, os desempenhos obtidos com o *driver* ativado foram ligeiramente inferiores (respectivamente 8,5% e 2,3% menor). Por fim, nos *benchmarks* Cinebench e 7zip, curiosamente, os desempenhos obtidos com o *driver* ativado foram ligeiramente superiores (4,3% e 1,6% maior). Em um primeiro momento, a hipótese produzida foi de que o driver estaria bloqueando grande parte dos processos do sistema, mas após realizados os logs dos processos bloqueados pelo driver durante as execuções dos *benchmarks*, foi observado que apenas dois executáveis foram bloqueados: um relacionado à máquina virtual (vm3dservice.exe) e outro relacionado ao sistema operacional (rundll32.exe). Diante dos resultados obtidos nos experimentos, não se pode afirmar que o funcionamento do *driver* implementado tenha causado custos computacionais significativos.

Tabela 2. Média de desempenho obtida para cada benchmark

Benchmark	Métrica	Driver Ativado	Média	Mediana	Overhead
Novabench	Pontos	Não	83,0	85,5	
Novabench	Pontos	Sim	83,0	82,5	0,0%
Aida64	Pontos	Não	3402,8	3394,5	
Aida64	Pontos	Sim	3074,2	3104,0	-8,5%
Cinebench	Pontos	Não	1092,4	1116,0	
Cinebench	Pontos	Sim	1145,2	1164,5	4,3%
Geekbench 6	Pontos	Não	1519,8	1521,5	
Geekbench 6	Pontos	Sim	1490,8	1486,5	-2,3%
7zip	Mips	Não	5160,6	5155,4	
7zip	Mips	Sim	5286,0	5241,0	1,6%

5.3. Efetividade do SAUCY SPICE com os *malwares* da APF

Pode ser observado pela Figura 5(b) que o modelo proposto teve uma precisão na identificação de $91,6\%$ quando utilizado sobre as amostras malignas relacionadas a APF, ou seja, 44 das 48 amostras poderiam ter sido identificadas.

5.4. Discussão

Os testes da Subseção 5.1 confirmaram a capacidade de o SAUCY SPICE gerar automaticamente regras mais genéricas, tendo atingido uma razão de 1 regra para representar 50 amostras. Embora a generalização de regras não permita uma eficácia de 100% de detecção, o parâmetro ϵ da solução pode ser ajustado para adequar relação falsos positivos (FP) / falsos negativos (FN) (e também o número de regras geradas) a uma necessidade em específico. Cabe ainda ressaltar que assinaturas manuais podem ser desenvolvidas e incorporadas à solução de modo a reduzir ainda mais os FP e FN residuais.

Em relação aos testes de desempenho, conforme mencionado na Subseção 5.2, não pôde ser observado *overhead* computacional causado pelo *driver* desenvolvido, evidenciando que se obteve êxito no desenvolvimento de um módulo de detecção leve. Entretanto,

trabalhos futuros ainda serão conduzidos no sentido de comparar diretamente o desempenho do *driver* ao das outras soluções existentes, que não funcionam totalmente em *kernel mode*. Além disso, pretende-se utilizar cenários de testes em que a base de assinaturas seja substancialmente maior do que a utilizada neste trabalho.

Cabe ressaltar que o treinamento da solução está limitado ao modo *offline* e em *batch*. Como a implementação do SAUCY SPICE utiza árvore de decisão, o uso de *online-learning* pode prejudicar a estrutura da árvore, na medida em que dados poucos representativos tendem a incorporar erros de decião a modelos dessa natureza. Ademais, devido ao fato de a solução realizar o processo de clusterização utilizando *imports* dos arquivos executáveis, espera-se que, caso estes *imports* sejam corrompidos por ofuscamento ou empacotamento *(malware packing)*, a solução não apresenterá o resultado desejado.

Por fim, a alta taxa de detecção obtida nos experimentos em cenário real evidenciam que a capacidade de generalização de regras da solução pode ser útil para evitar a ocorrência de crimes digitais. Aliando-se isso à proposta de baixa demanda computacional do SAUCY SPICE, essa solução pode vir a ser evoluída de modo a se tornar uma boa alternativa de proteção para os ativos de TI da APF.

6. Conclusões

Neste trabalho, apresentamos o SAUCY SPICE, uma solução para criação automatizada de assinaturas e bloqueio de ameaças de *malware* em SO Windows. A solução consiste em um processo automático de geração de políticas, que se mostraram eficientes e capazes de representar diversas amostras de *malware* por um reduzido conjunto de assinaturas. Concomitantemente, foi desenvolvido um módulo de detecção que funciona integralmente em *kernel mode*, com o objetivo final de minimizar o impacto no desempenho do sistema. Os resultados obtidos nos experimentos evidenciam que a solução é capaz de: gerar automaticamente assinaturas generalistas; realizar a detecção sem consumo relevante de recursos computacionais; melhorar a segurança cibernética da APF.

Trabalhos futuros podem ser conduzidos no sentido de: (a) avaliar a solução de modo a compará-la diretamente com outras já existentes; (b) implementar algoritmos e estruturas de dados mais eficientes no módulo de detecção; (c) realizar testes utilizando outras *features* em conjunto, inclusive dinâmicas; (d) avaliar a capacidade de generalização da solução em cenários mais desafiadores, como na presença de *zero-day malware*; (e) estender a implementação arquitetural do módulo de detecção para outros SO e avaliar o desempenho nestes; (f) incluir análise de tráfego de rede; e (g) realizar testes com outros algoritmos de machine learning de forma a aumentar a eficiência da solução na identificação de amostras maliciosas.

Agradecimentos

Apoiam financeiramente este trabalho o Programa de Pós-graduação em Aplicações Operacionais—PPGAO/ITA, a FAPESP processos #2020/09850-0, e o CNPq processo n.o 132058/2023-5.

Referências

Abusitta, A., Li, M. Q., and Fung, B. C. (2021). Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications*, 59:102828.

Aslan, Ö. A. and Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271.

Botacin, M., Alves, M. Z., Oliveira, D., and Grégio, A. (2022). Heaven: A hardware-enhanced antivirus engine to accelerate real-time, signature-based malware detection. *Expert Systems with Applications*, 201:117083.

Campion, M., Dalla Preda, M., and Giacobazzi, R. (2021). Learning metamorphic malware signatures from samples. *Journal of Computer Virology and Hacking Techniques*, 17(3):167–183.

David, O. E. and Netanyahu, N. S. (2015). Deepsign: Deep learning for automatic malware signature generation and classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Federal, P. (2021). Relatório do SISCRIM de crimes causados à Administração Pública Federal por ransonware. documento reservado e não publicado.

Feng, Y., Bastani, O., Martins, R., Dillig, I., and Anand, S. (2017). Automated synthesis of semantic malware signatures using maximum satisfiability. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'17)*, San Diego, CA.

Kaspersky (2022). Kaspersky Security Bulletin 2022. Statistics — securelist.com. `https://securelist.com/ksb-2022-statistics/108129/`. [Acessado em 20-Jun-2023].

Minghao, K., Chyang, K. Y., and Karuppiah, E. K. (2007). Performance analysis and optimization of user space versus kernel space network application. In *2007 5th Student Conference on Research and Development*, pages 1–6.

Mohanta, A. and Saldanha, A. (2020). *Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware*. Springer.

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69:066133.

Pinto, R., Delbem, A., and Monaco, F. (2017). Caracterização do perfil de consumo de recursos de programas binários utilizando a técnica damicore. In *Anais do XIII Simpósio Brasileiro de Sistemas de Informação*, Porto Alegre, RS, Brasil. SBC.

Razeghi Borojerdi, H. and Abadi, M. (2013). Malhunter: Automatic generation of multiple behavioral signatures for polymorphic malware detection. In *ICCKE 2013*.

Saitou, N. and Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425.

Sanches, A., Cardoso, J. M., and Delbem, A. C. (2011). Identifying merge-beneficial software kernels for hardware implementation. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 74–79.

Singh, J. and Singh, J. (2021). A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture*, 112:101861.

Tang, Y., Xiao, B., and Lu, X. (2009). Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms. *Computers & Security*, 28(8).