AutoDroid: disponibilizando a ferramenta DroidAugmentor como serviço

Luiz Felipe Laviola[1], **Kayuã Oleques Paim**[1], **Diego Kreutz**[1], **Rodrigo Brandão Mansilha**[1]

[1]Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

luiz@laviola.dev, {kayuapaim.aluno,diegokreutz,mansilha}@unipampa.edu.br

Resumo. *Propomos a AutoDroid, uma solução baseada em virtualização leve para disponibilizar como serviço a ferramenta DroidAugmentor, cujo objetivo é permitir o aumento de datasets utilizados para combater malwares Android através de IA. Disponibilizamos publicamente uma implementação da AutoDroid como prova de conceito. Apresentamos também uma avaliação qualitativa preliminar e esperamos que a solução viabilize a execução distribuída de outros serviços de AutoML, como a DroidAutoML.*

1. Introdução

O nível de proliferação de *malwares* é alarmante, pois atacantes têm utilizado massivamente técnicas sofisticadas de inteligência artificial (IA). Entre elas, destacamos os geradores inteligentes (p.ex., baseados em Redes Neurais Artificiais), que aprendem com aplicativos desonestos e honestos [Hu and Tan 2022]. Esses geradores ampliam em escala os desafios impostos às soluções de antivírus, pois podem usar aplicativos desonestos para explorar novas ameaças surgidas com o avanço tecnológico, bem como aplicativos honestos para camuflar aplicativos maliciosos.

Para enfrentar a emergência de *malwares* gerados por IA de maneira escalável, é imperativo compreender e adotar contramedidas baseadas em IA, como modelos preditivos [Meijin et al. 2022]. Porém, o sucesso dessas estratégias dependem significativamente da qualidade e quantidade dos conjuntos de dados de treinamento [AI & Data Today 2023]. Por isso, é crucial superar diversos desafios, como obsolescência dos dados existentes, escassez de amostras representativas, atrasos na rotulagem e falta de estratégias para identificação de mutações criadas automaticamente com IA. Diante disso, são necessários processos sistemáticos para criar, ajustar e evoluir *datasets* de *malware*, que não são triviais, como mostra a Figura 1.

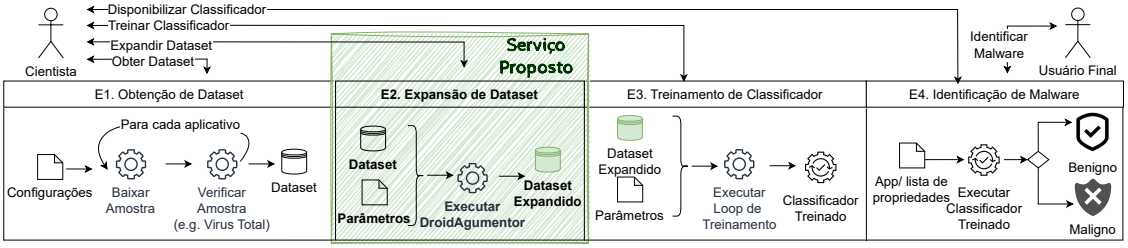

Figura 1. Pipeline de IA para detecção de *malware* com ênfase no serviço proposto.

A escassez de amostras representativas e a dificuldade em encontrar dados reais atuais (E1), especialmente em relação às mutações de *malware*

Android, tornam o problema de treinar bons classificadores (E3) desafiador [Miranda et al. 2022, Kouliaridis et al. 2020, Wang et al. 2019]. Um problema importante na detecção de *malwares* Android é a obsolescência, baixa qualidade e limitação dos *datasets* disponíveis, como diagnosticado em trabalhos recentes [Soares et al. 2021a, Soares et al. 2021b, Miranda et al. 2022]. Recentemente, propomos ferramentas como ADBuilder, AMGenerator e AMExplorer [Vilanova et al. 2022, Rocha et al. 2023] para a construção de *datasets* atualizados de maneira sistemática. Entretanto, devido a diversas limitações, como dificuldade de obtenção das amostras maliciosas e um processo de rotulação bastante lento[1], é tecnicamente inviável criarmos continuamente (p.ex., cada 2 ou 6 meses) conjuntos de dados atualizados para o treinamento e validação de modelos preditivos. Recentemente, temos desenvolvido também ferramentas para automatizar as etapas E3 e E4, como QuickAutoML [Siqueira et al. 2021] e DroidAutoML [Assolin et al. 2022].

Para realizar a Etapa 2, recentemente propomos a ferramenta DroidAugmentor [Casola et al. 2023], cujo objetivo é permitir o treinamento, a avaliação e a compreensão do funcionamento e dos desafios de utilização de *Generative Adversarial Networks* (GANs), mais especificamente *conditional* GANs (cGANs), para ampliação sintética de dados (p.ex., novas amostras de *malwares*). Resumidamente, o objetivo da DroidAugmentor é prover recursos técnicos para o pesquisador conseguir compreender o complexo processo de treinamento, avaliação e ajustes de cGANs para aumentar *datasets* de maneira a atingir melhores configurações da topologia, demonstrado nos classificadores e resultados nas etapas E3 e E4. Contudo, para executar a solução em escala adequada (isto é, considerando multitude de combinações de hiperparâmetros, variedades de *datasets*, número de repetições, épocas de treinamento, etc.) é necessário adequá-la para permitir sua execução em variados cenários, potencialmente composto por múltiplos pesquisadores e infraestrutura distribuída.

O objetivo geral deste trabalho é **propor uma solução para executar de maneira distribuída serviços como o DroidAugmentor** e, assim, conferir-lhe propriedades como escalabilidade, elasticidade e tolerância a falhas. Esperamos que a proposta possa fomentar a investigação de técnicas avançadas de geração de dados sintéticos para expandir *datasets* úteis na classificação de aplicativos Android (benigno ou maligno), como mostrado na Etapa E2 da Figura 1. Assim, apresentamos as seguintes contribuições técnicas para o estado-da-arte: (1) uma arquitetura baseada em micro-serviços para a execução de aplicações de processamento de *datasets*; e (2) instanciação da arquitetura utilizando como estudo de caso a ferramenta DroidAugmentor para expansão de *datasets* malware.

No restante deste trabalho, apresentamos a arquitetura na Seção 2, uma prova de conceito e avaliação preliminar na Seção 3, e considerações finais na Seção 4.

2. AutoDroid

Executar aplicações como a DroidAugmentor pode ser uma tarefa complexa, considerando requisitos como gerenciamento de dependências, escalabilidade e elasticidade.

[1]Por exemplo, apenas 250 rotulações por dia são possíveis utilizando a API do VirusTotal (`https://developers.virustotal.com/reference/overview`), o principal serviço existente para metadados de rotulação.

Encapsular essas aplicações em um esquema de virtualização pode ser suficiente para um cenário com um único usuário e/ou única máquina hospedeira, mas não é escalável para cenários complexos, multi-usuários e multi-máquinas, como é preciso para as diversas etapas do *pipeline* de IA mostrado na Figura 1.

A solução AutoDroid permite ao usuário executar aplicações como a DroidAugmentor de acordo com a arquitetura apresentada na Figura 2 com a notação C4[2]. Em resumo, o AutoDroid facilita a instanciação, o gerenciamento e a utilização de sistemas de processamento de *datasets* (componente Data Processor), como o DroidAugmentor, por usuários interessados em análise de dados (User). É importante destacarmos que consideramos o seguinte modelo de ameaças: (a) assumimos como seguras primitivas criptográficas como funções de *hash* (e.g., SHA256) e MAC (e.g., HMAC) e protocolos como TLS 1.3; (b) assumimos um ambiente de execução seguro (e.g., *datacenter* ou nuvem privada), evitando a necessidade de protocolos como TLS entre os elementos internos do sistema; (c) assumimos que os administradores do sistema e da infraestrutura são confiáveis, isto é, não há *insider threats*; e (d) assumimos que a aplicação (e.g., DroidAugmentor) é confiável, isto é, foi auditada e não traz consigo ameaças ao ambiente (e.g., *backdoor*, *malware*).

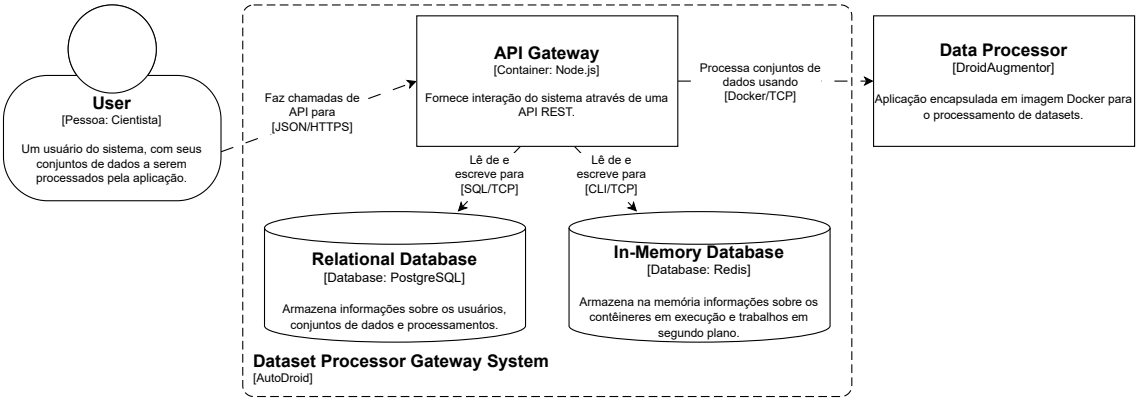

Figura 2. Arquitetura da AutoDroid.

Os principais componentes da AutoDroid são os seguintes. O Data Processor abstrai a aplicação desejada (p.ex., DroidAugmentor) encapsulando-a em uma tecnologia de virtualização leve (p.ex., Docker) devidamente configurada (p.ex., padrão de entrada e saída, domínio de valores aceitáveis para cada parâmetro de entrada). Em síntese, a AutoDroid atua como gerenciador/orquestrador de instâncias de Data Processor. Essa arquitetura confere flexibilidade ao sistema: o administrador do sistema pode agregar novos processadores de *datasets* (ou diferentes versões do mesmo) simplesmente especificando uma imagem existente do repositório (p.ex., Docker Hub[3]) no arquivo de configuração (p.ex., usando o formato de intercâmbio de dados JSON). Esse arquivo é carregado na inicialização da aplicação e serve para definir os processadores disponíveis, suas configurações e a imagem necessária, que é carregada automaticamente na inicialização da API. O API Gateway é o elemento central do AutoDroid, expondo uma API REST[4] conectada a um banco de dados relacional

[2] `https://c4model.com/`
[3] `https://hub.docker.com/`
[4] `https://restfulapi.net/`

(p.ex., PostgreSQL[5]) e a um banco de dados baseado em memória (p.ex., Redis[6]).

A Figura 3 apresenta o fluxograma de execução principal da AutoDroid. Inicialmente são realizados testes e tomadas medidas necessárias para criação de usuário e *dataset* de entrada. A chamada de processamento é assíncrona considerando que a etapa ter durações muito variadas, na ordem de minutos, horas ou dias. O sistema pode consultar periodicamente a situação até que o processamento seja concluído. Após a execução é possível obter o resultado (sucesso ou falha) e os arquivos de saída, como *dataset* sintético e gráficos de qualidade do *dataset* da DroidAugmentor.

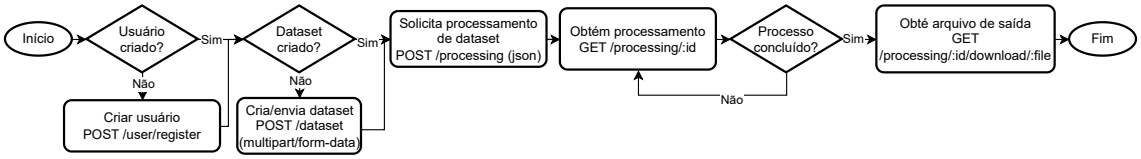

Figura 3. Fluxograma de execução da AutoDroid.

Como delimitação de escopo de projeto, consideramos duas simplificações principais. A primeira é relacionada ao esquema de permissões e gerenciamento de usuários. Atualmente, o usuário, anônimo, é criado e mantido automaticamente de maneira transparente. Portanto, a camada de gerenciamento seguro de usuários e grupos, integrada com plataformas de autenticação e autorização (p.ex., diretórios LDAP e AD ou ainda sistemas de federação como a CAFe) é deixada para trabalhos futuros. Como segunda delimitação, propomos um esquema simplificado para acompanhamento da etapa de processamento com apenas quatro estágios: preparado, executando, pronto e falho. Futuramente, pretendemos agregar controles avançados, especialmente no contexto de treinamento de redes neurais (p.ex., épocas de treinamento atual e total), e também de verificação de uso de recursos para otimizar eficiência no uso de recursos.

3. Avaliação Preliminar

Nesta seção, apresentamos uma implementação como prova de conceito e uma análise qualitativa preliminar da solução proposta.

3.1. Implementação e Ambiente de Testes

Como prova de conceito, desenvolvemos e tornamos pública uma versão da AutoDroid[7] completamente funcional e documentada. Executamos com sucesso o teste funcional básico em dois ambientes distintos[8]. A implementação fez uso das seguintes tecnologias primárias: Node.js como gateway para o serviço web, PostgreSQL para persistência de dados estruturados e Redis para armazenamento em memória. O gerenciamento de virtualização leve é realizado com tecnologia Docker. Essas escolhas tecnológicas foram fundamentadas na popularidade, maturidade e suporte

[5] `https://www.postgresql.org/`

[6] `https://redis.io/`

[7] `https://github.com/luizfelipelaviola/autodroid` (*tag wrseg23*

[8] (1) Host Ubuntu 22.04, e (2) OS WSL: Ubuntu 22.04.2, OS Host: Windows 10 Pro 22H2

pela comunidade de desenvolvedores e o conhecimento técnico da equipe envolvida no projeto.

Para demonstrar as potencialidades da AutoDroid, comparamos quatro ambientes possíveis (A1-A4) para disponibilizar a DroidAgumentor. No primeiro ambiente, denominado A1, o serviço é mantido (isto é, download, instalação, manutenção de dependências) e executado nativamente na máquina hospedeira, ou seja, sem apoio da AutoDroid. No Ambiente A2, o serviço é instanciado em um ambiente virtualizado leve pré-configurado, mas também sem apoio da AutoDroid. No Ambiente A3, a AutoDroid é configurada na máquina hospedeira e executa o DroidAugmentor como serviço em um ambiente de virtualização leve. No Ambiente A4, a AutoDroid é configurado em ambiente virtualizado e executa a DroidAugmentor em ambiente virtualizado (*i.e.*, Docker *in* Docker).

Para cada ambiente, nós listamos as principais dependências tecnológicas que precisam ser instaladas na máquina subjacente. Com base na quantidade e complexidade das dependências, nós avaliamos subjetivamente dois critérios considerando uma escala entre 1 (melhor ambiente) e 4 (pior ambiente). O primeiro critério, nível de **sobrecarga**, é sobre o custo computacional (temporal e espacial) adicionais que cada ambiente exige. O segundo critério, **manutenibilidade**, é relacionado à dificuldade de se configurar, utilizar e atualizar o ambiente. O conjunto de cenários resulta da combinação entre singularidade ou pluralidade de usuários e máquinas hospedeiras. Para cada cenário, nós analisamos prós (manutenabilidade) e contras (sobrecarga) e apresentamos uma recomendação seguindo a mesma escala anterior.

3.2. Análise

A Tabela 1 apresenta os resultados da nossa comparação preliminar e sintetiza as limitações e potencialidades da AutoDroid. Em resumo, para o cenário onde um pesquisador solo usa uma máquina dedicada para a DroidAugmentor, provavelmente o ambiente A1 é preferível do ponto de vista de sobrecarga, já que a manutenibilidade tende a ser menos relevante. Por outro lado, para um grupo trabalhando em equipe visando testar e evoluir métodos de IA em escala, isto é, potencialmente usando múltiplos ambientes em nuvem, recomendamos usar a AutoDroid de maneira virtualizada. Entre os dois limiares há opções intermediárias que devem servir para casos particulares.

Tabela 1. Comparação entre ambientes e cenários para executar DroidAugmentor.

#	Ambiente de execução	Dependências	Sobrecarga	Manutenabilidade	Hospedeiro único		Hospedeiros múltiplos	
					Usuário único Recomendação	Usuário múltiplo Recomendação	Usuário único Recomendação	Usuário múltiplo Recomendação
A1	DroidAugmentor em ambiente nativo	Python e bibliotecas	1	4	1	3	4	4
A2	DroidAugmentor em ambiente virtual	Docker	2	3	2	1	3	3
A3	DroidAugmentor via AutoDroid em ambiente nativo	Docker, Node.js, Yarn	3	2	3	2	1	2
A4	DroidAugmentor via AutoDroid em ambiente virtual	Docker	4	1	4	4	2	1

4. Considerações Finais

Neste trabalho apresentamos a AutoDroid: uma solução para executar o DroidAugmentor como serviço através de virtualização leve. Apresentamos uma implementação disponível publicamente como prova de conceito e uma avaliação qualitativa para comparar a proposta com outros ambientes considerando variados cenários

derivados da combinação entre um ou múltiplos usuários realizando avaliação em uma ou múltiplas máquinas hospedeiras.

Esperamos que AutoDroid possa impactar positivamente na investigação e disponibilização de *datasets malware*, facilitando a execução de processos de maneira distribuída para conferir escalabilidade, elasticidade e tolerância a falhas. Nessa direção, este trabalho é um primeiro passo. Como trabalhos futuros, pretendemos avaliar quantitativamente os ganhos permitidos pela solução em comparação com as alternativas de execução. Futuramente, esperamos que a solução permita executar como serviço *pipelines* complexos de AutoML, como os implementados por ferramentas como DroidAutoML e QuickAutoML.

Agradecimentos. Este trabalho foi apoiado pela FAPERGS (TO 22/2551-0000841-0) e CAPES – Código de Financiamento 001.

Referências

AI & Data Today (2023). Top 10 reasons why ai projects fail. `https://t.ly/wMBj5`.

Assolin, J., Kreutz, D., Siqueira, G., Rocha, V., Miers, C., Mansilha, R., and Feitosa, E. (2022). DroidAutoML: uma ferramenta de automl para o domínio de detecção de malwares android. In *Anais Estendidos do XXII SBSeg*, pages 135–142.

Casola, K., Paim, K., Mansilha, R., and Kreutz, D. (2023). Droidaugmentor: uma ferramenta de treinamento e avaliação de cgans para geração de dados sintéticos. In *Anais Estendidos do XXIII SBSeg*. SBC.

Hu, W. and Tan, Y. (2022). Generating adversarial malware examples for black-box attacks based on GAN. In *International Conference on Data Mining and Big Data*, pages 409–423. Springer.

Kouliaridis, V., Kambourakis, G., and Peng, T. (2020). Feature importance in android malware detection. In *IEEE 19th TrustCom*, pages 1449–1454. IEEE.

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., Yinwei, W., and Jiaxuan, G. (2022). A systematic overview of android malware detection. *Applied Artificial Intelligence*, 36(1):2007327.

Miranda, T. C., Gimenez, P.-F., Lalande, J.-F., Tong, V. V. T., and Wilke, P. (2022). Debiasing android malware datasets: How can i trust your results if your dataset is biased? *IEEE Transactions on Information Forensics and Security*, 17:2182–2197.

Rocha, V., Assolin, J., Bragança, H. L., Kreutz, D., and Feitosa, E. (2023). Amgenerator e amexplorer: Geração de metadados e construção de datasets android. In *Anais Estendidos do XXIII SBSeg*. SBC.

Siqueira, G., Rodrigues, G., Feitosa, E., and Kreutz, D. (2021). Quickautoml: Uma ferramenta para treinamento automatizado de modelos de aprendizado de máquina. In *Anais da XIX Escola Regional de Redes de Computadores*, pages 85–90. SBC.

Soares, T., Mello, J., Barcellos, L., Sayyed, R., Siqueira, G., Casola, K., Costa, E., Gustavo, N., Feitosa, E., and Kreutz, D. (2021a). Detecção de malwares android: Levantamento empírico da disponibilidade e da atualização das fontes de dados. In *Anais da XIX ERRC*, pages 49–54. SBC.

Soares, T., Siqueira, G., Barcellos, L., Sayyed, R., Vargas, L., Rodrigues, G., Assolin, J., Pontes, J., Feitosa, E., and Kreutz, D. (2021b). Detecçao de malwares android: datasets e reprodutibilidade. In *Anais da XIX ERRC*, pages 43–48. SBC.

Vilanova, L., Kreutz, D., Assolin, J., Quincozes, V., Miers, C., Mansilha, R., and Feitosa, E. (2022). Adbuilder: uma ferramenta de construçao de datasets para detecçao de malwares android. In *Anais Estendidos do XXII SBSeg*, pages 143–150. SBC.

Wang, H., Si, J., Li, H., and Guo, Y. (2019). RmvDroid: Towards a reliable android malware dataset with app metadata. In *IEEE/ACM MSR*, pages 404–408.