XXI Escola Regional de Redes de Computadores / VIII Workshop de Segurança (ERRC/WRSeg). DOI: 10.5753/errc.2024.4679

ARTIGO COMPLETO/FULL PAPER

MH-API: Uma Solução Escalável Para Detecção de Malwares em Android

MH-API: A Scalable Solution for Detecting Malwares on Android

Joner de Mello Assolin • joner.assolin@icomp.ufam.edu.br
Universidade Federal do Amazonas (UFAM)

Hendrio Bragança • hendrio.luis@icomp.ufam.edu.br
Universidade Federal do Amazonas (UFAM)

Diego Kreutz • diegokreutz@unipampa.edu.br
Universidade Federal do Pampa (UNIPAMPA)

Eduardo Feitosa • efeitosa@icomp.ufam.edu.br
Universidade Federal do Amazonas (UFAM)

Sávio Saboia • savio.saboia@icomp.ufam.edu.br
Universidade Federal do Amazonas (UFAM)

Luiza Leão • lpml@icomp.ufam.edu.br
Universidade Federal do Amazonas (UFAM)

Vanderson Rocha • vanderson@ufam.edu.br
Universidade Federal do Amazonas (UFAM)

RESUMO. A MH-API foi desenvolvida para análise de aplicativos Android e criação de *datasets* em larga escala. Utiliza um cache local com metadados de mais de 2 milhões de APKs, acelerando a recuperação de dados e melhorando a escalabilidade. Integrada ao VirusTotal para análises complementares e ao AndroZoo para obtenção de APKs ausentes, amplia a cobertura de amostras disponíveis. A API também inclui um endpoint com modelo preditivo para classificação de aplicativos, aumentando a precisão na detecção de ameaças. Projetada para flexibilidade, permite integração com outros sistemas, otimizando fluxos de trabalho e facilitando atualizações contínuas em um ambiente centralizado.

ABSTRACT. The MH-API was developed for analyzing Android applications and creating large-scale datasets. It uses a local cache with metadata from over 2 million APKs, accelerating data retrieval and enhancing scalability. Integrated with VirusTotal for complementary analyses and AndroZoo for retrieving missing APKs, it broadens the available sample coverage. The API also includes an endpoint with a predictive model for application classification, improving threat detection accuracy. Designed for flexibility, it allows integration with other systems, optimizing workflows and enabling continuous updates in a centralized environment.

PALAVRAS-CHAVE: Malwares Android • MH-API • Aprendizado de Maquinas • Conjunto de Dados • Virustotal
KEYWORDS: Malwares Android • MH-API • Machine Learning • Dataset • Virustotal

1 Introdução

A criação de *datasets* atualizados e de alta qualidade continua sendo um desafio crucial na segurança cibernética [1–5]. Os conjuntos de dados amplamente utilizados apresentam limitações, como desatualização, insuficiência de características e marcação incorreta de amostras [6, 7], comprometendo a precisão dos modelos de *machine learning*, essenciais para identificar novos malwares no Android.

Com a crescente sofisticação dos ataques, a demanda por *datasets* representativos e atualizados aumenta, pois esses modelos precisam acompanhar a evolução das ameaças para manter sua eficácia. Uma prática comum para rotulação de amostras é o uso do VirusTotal[1], que agrega resultados de múltiplos *scanners* antivírus. Contudo, a capacidade limitada de requisições e o uso de dados em cache dificultam análises em larga escala e atualizações frequentes, conforme observado no AMGenerator [8], onde amostras inicialmente benignas foram posteriormente classificadas como maliciosas.

Nesse cenário, um grande desafio é desenvolver *datasets* em larga escala que permitam uma análise eficiente de grandes volumes de APKs, superando as limitações do VirusTotal. Para atender à natureza dinâmica dos malwares e à necessidade crescente de dados em tempo real, é essencial uma solução que possibi-

[1] https://www.virustotal.com/

lite a coleta massiva de metadados, com análises mais precisas e escaláveis.

A ferramenta proposta para enfrentar esses desafios é a MH-API, desenvolvida para superar as limitações do VirusTotal e oferecer uma infraestrutura escalável e eficiente. A MH-API permite a criação rápida de *datasets* por meio de um cache local que armazena metadados de mais de 2 milhões de APKs e está integrada ao repositório AndroZoo[2], facilitando a obtenção de amostras ausentes. Para contornar as restrições de requisições do VirusTotal, a MH-API utiliza múltiplas chaves de API, viabilizando análises paralelas em larga escala. Além disso, a API incorpora modelos preditivos especializados em malwares Android, aumentando a precisão na detecção de ameaças. Sua arquitetura flexível permite integração automatizada com outros sistemas, otimizando o fluxo de trabalho e promovendo um ambiente mais dinâmico e eficaz.

2 Visão Geral da MH-API

A arquitetura apresentada na Figura 1 ilustra uma solução centrada em uma API que atua como intermediária entre o cliente e diversos serviços de *backend* especializados. O objetivo principal dessa API é processar e analisar aplicativos Android, utilizando informações fornecidas pelo cliente, que podem ser um APK completo ou o *hash* SHA256 de um arquivo. A API gerencia essas solicitações, distribuindo-as para diferentes serviços, responsáveis por tarefas como extração de metadados, verificação de segurança, download de arquivos e análise preditiva. Essa estrutura modular e distribuída permite que cada serviço execute suas funções de forma independente, garantindo escalabilidade e eficiência no processamento das solicitações.

No componente de **Client (Cliente)**, o usuário interage diretamente com a API por meio de uma série de *endpoints*, como **mh_ag_metadata, mh_vt_metadata, mh_ag_features, mh_predict, mh_analysis, mh_az_download, mh_search_cache**, detalhados na Tabela 1. Através desses *endpoints*, o cliente pode enviar um APK ou um *hash* SHA256, e a API roteia a solicitação para o serviço adequado. Esses *endpoints* cobrem todo o espectro de funcionalidades, desde a extração de metadados estáticos até a busca no cache por amostras armazenadas ou o download de novos APKs.

A API funciona como o núcleo dessa arquitetura, coordenando as interações entre o cliente e os serviços de *backend*. Sua principal função é processar as requisições enviadas pelo cliente e distribuir as tarefas entre serviços especializados que realizam, por exemplo, a extração de informações estáticas dos APKs, verificação de segurança com fontes externas (como o VirusTotal) e análise preditiva de comportamentos maliciosos. Além disso, a API consulta um cache local para otimizar as respostas, reduzindo a dependência de serviços externos sempre que os dados já estiverem disponíveis.

Os Serviços de *Backend* incluem várias integrações essenciais para o fluxo de trabalho. O AndroGuard[3] Service é utilizado para análise estática, permitindo a extração de características como permissões, atividades e *intents* diretamente do APK. O VirusTotal Service verifica se o APK já foi analisado anteriormente quanto ao potencial de malware, e, se os dados estiverem disponíveis, eles são retornados ao cliente. O AndroZoo Service é responsável por baixar APKs, principalmente para alimentar o cache local. Já o ML (Machine Learning) Service aplica modelos preditivos treinados para avaliar se um APK é benigno ou malicioso. Por fim, o Cache local armazena e recupera dados processados anteriormente, acelerando as respostas e minimizando consultas externas desnecessárias.

O Fluxo de Trabalho inicia quando o cliente envia um APK ou um SHA256 para a API. Em primeiro lugar, a API verifica se as informações já estão no cache. Se não estiverem, as requisições são distribuídas para os serviços apropriados. O AndroGuard extrai as características estáticas, o VirusTotal consulta informações de segurança, o AndroZoo baixa amostras de APKs, e o serviço de aprendizado de máquina realiza predições sobre o comportamento do aplicativo. O resultado final é um conjunto consolidado de informações que é retornado ao cliente no formato JSON, incluindo metadados, características e predição.

Essa arquitetura foi desenvolvida com o propósito de automatizar a análise de APKs, combinando diversas fontes de informações e métodos de avaliação, como consultas ao cache local, busca por malware no VirusTotal e predições baseadas em aprendizado de máquina. Isso garante que o sistema ofereça uma análise eficiente e abrangente, auxiliando na classificação precisa de aplicativos Android, especialmente no contexto de detecção de malware.

A arquitetura da MH-API foi projetada para automatizar a análise de APKs, integrando diversas fontes de informação e métodos de avaliação, como consultas ao cache local, verificações de malware no VirusTotal e predições baseadas em modelos de aprendizado de máquina. Essa integração permite que o sistema realize uma análise mais detalhada e eficiente, facilitando

[2] https://androzoo.uni.lu/

[3] https://androguard.readthedocs.io/

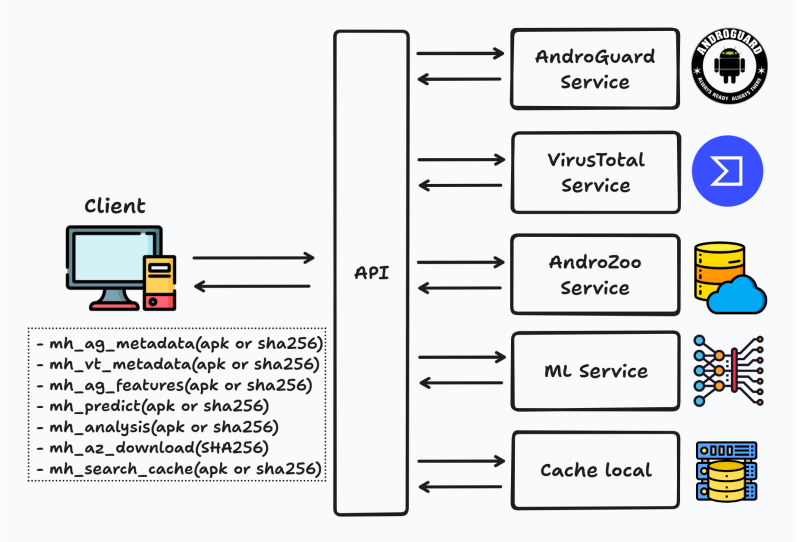

Figura 1. Visão geral da MH-API e seus serviços.

a classificação de aplicativos Android, especialmente no contexto da detecção de malwares. Além disso, a arquitetura modular e escalável da MH-API a posiciona como uma solução poderosa e versátil para enfrentar as crescentes ameaças no ecossistema Android.

3 Avaliação da API

Para avaliar o desempenho da API, foi utilizado o *endpoint* `mh_analysis`, ilustrado na Figura 2, responsável por integrar diversos *endpoints* e fornecer um conjunto completo de dados, incluindo características e metadados extraídos do AndroGuard e do VirusTotal. Esses dados são obtidos tanto a partir da cache local quanto de serviços externos, caso as informações não estejam disponíveis na cache.

O teste de desempenho foi realizado por meio de um *script* em Python que automatizou 1000 requisições, mensurando a média do tempo de execução, o consumo de memória RAM e o espaço em disco necessário para o processamento. O ambiente de teste consistiu em um notebook com as seguintes especificações: Hardware: Intel(R) Core(TM) i7-1185G7, 32GB de RAM. Software: Windows 11 Pro, compilação 22631.3880, Python 3.9.13.

3.1 Cenário 1: dados disponíveis na cache local

O primeiro cenário de teste tem como objetivo avaliar o desempenho do *endpoint* `mh_analysis` quando as informações solicitadas já estão previamente armazenadas na cache local. Esse cache contém mais de 2 milhões de registros no formato JSON, abrangendo dados relevantes, como características (*features*) associadas a cada aplicativo, a classificação (*label*) baseada no número de varreduras pelo VirusTotal, a data da última análise e metadados como nome do pacote e versão do aplicativo. Além disso, as predições do modelo são baseadas em uma probabilidade que varia de 0 a 1, indicando a confiança do modelo na classificação de um APK como malicioso. Durante o processo de análise, o modelo calcula a probabilidade de um APK pertencer à classe positiva (malware). Caso essa probabilidade seja maior ou igual a um limiar pré-definido, que por padrão é 0,5, o APK é classificado como malicioso.

Ao realizar uma solicitação ao *endpoint* com uma lista de 1.000 *hashes* SHA256, o sistema deve retornar o JSON correspondente contendo todas essas informações, demonstrando que a cache local está sendo corretamente utilizada. Esse cenário simula um caso de uso típico, no qual os dados já estão disponíveis para acesso eficiente, sem a necessidade de consultas a serviços externos.

Os resultados obtidos durante o teste indicam que um total de 1.000 *hashes* SHA256 foram processados em 130,47 segundos. A memória total consumida foi de 277,45 MB, enquanto o espaço em disco ocupado pela cache foi de 268 MB. Esses resultados confirmam a eficiência do sistema ao operar em cenários onde os dados estão disponíveis localmente, fornecendo respostas rápidas e minimizando o uso de recursos computacionais.

3.2 Cenário 2: recuperação de dados via serviços externos

O segundo cenário de teste foca em uma situação desafiadora, na qual os dados solicitados não estão disponíveis na cache local e também não há registros de

Tabela 1. Endpoints da MH-API.

Endpoint	**Método**	**Entrada**	**Saída**
mh_search_cache	`GET`	SHA256 ou APK	True ou False
Descrição: Pesquisa no cache por um SHA256 ou APK. **Uso:** Verificar se o dado já está armazenado localmente.			
mh_ag_metadata	`POST`	SHA256 ou APK	Metadados
Descrição: Retorna metadados de um APK ou SHA256, processando com o Androguard se não estiver no cache. **Uso:** Geração de metadados detalhados (nome, versão, pacote, etc.).			
mh_ag_features	`POST`	SHA256 ou APK	Características
Descrição: Extrai características (permissões, intents, etc.) do APK. **Uso:** Extração automática para análise e modelos de machine learning.			
mh_vt_metadata	`POST`	SHA256 ou APK	Resultados do VirusTotal
Descrição: Consulta o VirusTotal para dados de análise. **Uso:** Fornecer uma análise abrangente utilizando várias engines e scanners de segurança.			
mh_analysis	`POST`	SHA256 ou APK	Rótulo, características, predição
Descrição: Faz análise completa com base em cache, VirusTotal e AndroZoo. **Uso:** Consolidar dados para geração de datasets e predições.			
mh_az_download	`POST`	Lista de SHA256s	APKs baixados
Descrição: Baixa APKs do AndroZoo com base nos SHA256s fornecidos. **Uso:** Automação de downloads de APKs.			
mh_predict	`POST`	APK	Predição
Descrição: Realiza predição do comportamento do APK (malicioso ou benigno). **Uso:** Avaliação de novos APKs com modelos de machine learning.			

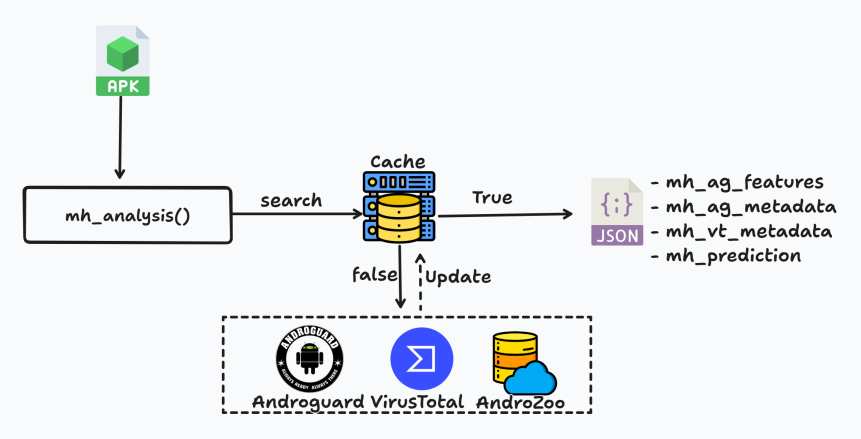

Figura 2. Endpoint **mh_analysis**.

análises anteriores no VirusTotal. Nesse contexto, o *endpoint* `mh_analysis` deve buscar as informações necessárias em fontes externas.

Quando um *hash* SHA256, cujo dados não estão disponíveis no cache local, é fornecido, a API inicia o processo de download do APK correspondente por meio do serviço AndroZoo. Neste teste, o aplicativo baixado tinha um tamanho de 25 MB.

Após o download, o APK é enviado ao VirusTotal para análise, enquanto, em paralelo, a ferramenta realiza a extração local de características e metadados utilizando o *AMGenerator*. Os dados extraídos, que incluem informações essenciais sobre o aplicativo, são então processados por um modelo de aprendizado de máquina (ML) para gerar uma predição sobre o comportamento do aplicativo, determinando se ele é malicioso ou benigno. Ao término da análise, as informações são armazenadas na cache local para consultas futuras, e o APK baixado é removido, liberando espaço em disco.

Durante este teste, foram utilizadas 3 amostras de APKs de tamanhos distintos: 553 KB, 25.387 KB e 72.323 KB. Cada amostra foi executada 10 vezes, totalizando 30 execuções. Os tempos de execução foram decompostos para cada serviço e processo, como: busca no cache local, download do APK, extração de features e metadados, além da análise no VirusTotal.

Os resultados na Tabela 2 mostram variações significativas no tempo de processamento entre arquivos APK, evidenciando que o tamanho do arquivo impacta diretamente o desempenho. O APK com SHA256 0D69E420F7F6EAEC48AA0C7DA5E05E91B94245DC4731E84AA7C583CBED694C0B, com 553 KB, teve o me-

SHA256	Tamanho	Tempo						
		mh_az_download	mh_ag_features	mh_ag_metadata	mh_vt_metadata	mh_predict	Total	Desvio Padrão
0D69E420F7F6EAEC48AA0C7DA5E05E91B94245DC4731E84AA7C583CBED694C0B	553 KB	7.35s	3.21s	2.78s	0.97s	3.47s	17.78s	2.65s
E1A00A456F8106E8A16FBA67F92DCD6B0E3ACF89C050EC656846451D63B79068	25.387 KB	301.82s	50.09s	42.50s	0.99s	50.60s	446.00s	29.92s
E19819F5685B507208D1E78853CFAE58157A074149AE7F333D8AFEC3F6D2E682	72.323 KB	912.22s	150.54s	148.20s	0.98s	143.44s	912.22s	22.00s

Tabela 2. Tabela de Resultados Médios por SHA256

nor tempo médio (17,78 segundos). Em contraste, o APK de 25,387 KB (SHA256 E1A00A456F8106E8A16FBA67F92DCD6B0E3ACF89C050EC656846451D63B79068) levou 446 segundos, refletindo o impacto do tempo de download e processamento. O APK maior, com 72,323 KB (SHA256 E19819F5685B507208D1E78853CFAE58157A074149AE7F333D8AFEC3F6D2E682), demorou 912,22 segundos, principalmente devido ao longo tempo de download.

O desvio padrão dos tempos também variou: o APK menor teve um desvio padrão de 2,65 segundos, indicando estabilidade, enquanto arquivos maiores apresentaram desvios maiores, como 29,92 segundos para o APK de 25,387 KB, possivelmente devido a variações na rede e no processamento. Isso sugere que arquivos maiores introduzem maior variabilidade, especialmente em etapas dependentes de recursos externos.

A média de tempo para 30 execuções foi de 609,8 segundos (10 minutos e 10 segundos), com desvio padrão de 42,34 segundos e consumo de memória de 103 MB. Esses resultados indicam que, mesmo sem dados armazenados previamente, a ferramenta integra eficientemente serviços externos para uma análise completa e robusta.

3.3 Cenário 3: 100% de cache local *miss*

Com base nos resultados do segundo cenário, estimou-se o desempenho do sistema ao processar 1.000 amostras com 100% de *miss* na cache, ou seja, quando todos os dados são obtidos de serviços externos. Esse é o pior caso, consumindo mais tempo e recursos.

O tempo total estimado para processar 1.000 amostras seria aproximadamente 459.000 segundos (127,5 horas), refletindo o elevado custo temporal das consultas externas e do processamento completo das amostras. O consumo total de memória seria de cerca de 982 MB, indicando uma eficiência aceitável, mesmo com o grande volume de dados. O espaço em disco necessário durante o processamento dos APKs é estimado em cerca de 820 MB, um volume modesto e facilmente gerenciável diante da capacidade de armazenamento dos sistemas atuais, sendo eficiente, mesmo em ambientes com recursos limitados, como servidores em nuvem.

Essas estimativas destacam o impacto do processamento em larga escala na API. O tempo elevado para amostras ausentes na cache local evidencia a necessidade de otimizações, especialmente quando há forte dependência de serviços externos, como VirusTotal e AndroZoo. Reduzir essa dependência, com pré-carregamento de dados ou melhorias na eficiência da cache, pode melhorar significativamente o desempenho, tornando o sistema mais ágil e responsivo em cenários de alto volume de requisições.

4 Conclusão

Este trabalho apresentou a MH-API, uma ferramenta eficiente para análise em larga escala de aplicativos Android, focada na detecção de malwares e construção de *datasets* robustos. Integrando serviços como cache local, AndroZoo, VirusTotal e modelos de aprendizado de máquina, a MH-API permite análises rápidas e precisas.

Nos testes, o sistema processou 1.000 amostras com eficiência quando os dados estavam em cache, mas apresentou maior tempo de processamento ao acessar fontes externas, evidenciando a importância de uma cache atualizada.

A API demonstrou ser escalável, com uso eficiente de memória e potencial para aprimorar a detecção de malwares e a criação de *datasets* de alta qualidade. Futuras melhorias podem focar na redução de tempo em cenários sem cache e na otimização de recursos para grandes volumes de dados.

Declarações complementares

Financiamento

Esta pesquisa foi parcialmente financiada, conforme previsto nos Arts. 21 e 22 do Decreto No. 10.521/2020, nos termos da Lei Federal No. 8.387/1991, através do convênio No. 003/2021, firmado entre ICOMP/UFAM, Flextronics da Amazônia Ltda e Motorola Mobility Comércio de Produtos Eletrônicos Ltda. O presente trabalho foi realizado também com apoio da CAPES – Código de Financiamento 001 e da FAPERGS, através dos termos de outorga 24/2551-0001368-7 e 24/2551-0000726-1.

Referências

1 Yavanoglu, O.; Aydos, M. A Review on Cyber Security Datasets for Machine Learning Algorithms. *In*: IEEE. 2017 IEEE international conference on big data (big data). 2017. P. 2186–2193.

2 Zheng, M. *et al.* Cybersecurity Research Datasets: Taxonomy and Empirical Analysis. *In*: 11TH USENIX Workshop on Cyber Security Experimentation and Test (CSET 18). 2018.

3 Nilă, C.; Patriciu, V.; Bica, I. Machine Learning Datasets for Cyber Security Applications. *Security & Future*, Scientific Technical Union of Mechanical Engineering"Industry 4.0", v. 3, n. 3, p. 109–112, 2019.

4 Alshaibi, A. *et al.* The Comparison of Cybersecurity Datasets. *Data*, MDPI, v. 7, n. 2, p. 22, 2022.

5 Bragança, H. *et al.* Capturing the Behavior of Android Malware with MH-100K: A Novel and Multidimensional Dataset. *In*: SBC. ANAIS do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. 2023. P. 510–515.

6 Wang, H. *et al.* RMVDroid: Towards a Reliable Android Malware Dataset With App Metadata. *In*: IEEE. 2019 IEEE/ACM 16th international conference on mining software repositories (MSR). 2019. P. 404–408.

7 Soares, T.; Siqueira, G.; Barcellos, L. *et al.* Detecção de Malwares Android: Datasets e Reprodutibilidade. *In*: ANAIS da XIX ERRC. Charqueadas/RS: SBC, 2021. P. 43–48. DOI: 10.5753/errc.2021.18540. Disponível em: https://sol.sbc.org.br/index.php/errc/article/view/18540.

8 Rocha, V. *et al.* AMGenerator e AMExplorer: Geração de Metadados e Construção de Datasets Android. *In*: SBC. ANAIS Estendidos do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. 2023. P. 41–48.