Freki: Uma Ferramenta para Análise Automatizada de *Malware*

Cristian H. M. Souza[1], **Felipe S. Dantas Silva**[1]

[1]LaTARC Research Lab
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN)
Natal-RN, Brasil

cristianmsbr@gmail.com, felipe.dantas@ifrn.edu.br

Abstract. *Malware analysis is of utmost importance for information security. However, while many tools attempt to automate this process, few have a local installation option and a centralized environment for analyzing and exposing results. Furthermore, API query limitations and high license prices make these tools unattractive for newcomers to the field. This work proposes* `Freki`*, an open-source tool for automated analysis of malicious programs. The solution features a simple and intuitive interface in which users can submit files for analysis and promptly obtain the results obtained by the system. In addition, the proposed system delivers a REST API that allows new analyzes and queries to already investigated binaries via hash, allowing* `Freki` *to be the basis for the development of new applications for malware analysis.*

Resumo. *A análise de malware é de fundamental importância para a segurança da informação. Embora diversas ferramentas tentem automatizar esse processo, poucas apresentam uma opção de instalação local e um ambiente centralizado para análise e exposição dos resultados. Ademais, as limitações de consultas a APIs e os altos preços de licenças de uso tornam essas ferramentas pouco atrativas para iniciantes na área. Este trabalho propõe o* `Freki`*, uma ferramenta open-source para análise automatizada de programas maliciosos. A solução apresenta uma interface simples e intuitiva, na qual os usuários podem enviar arquivos para análise e obter prontamente os resultados obtidos pelo sistema. Além disso, a solução possui uma API REST que permite novas análises e consultas a binários já analisados via hash, permitindo que o* `Freki` *seja base para o desenvolvimento de novas aplicações para análise de malware.*

1. Introdução

Infecções por *malware* continuam sendo uma das principais ameaças aos sistemas computacionais. Tais programas são responsáveis por causar prejuízos, através de alterações indevidas ou roubo de informações de usuários domésticos e empresas. Uma das formas eficazes de minimizar eventuais acessos não autorizados e, consequentemente, manter a integridade das informações é o uso da classificação via código *hash* [Kaur and Nagpal 2012]. Entretanto, essa técnica se torna ineficaz no combate de ameaças do tipo *zero-day*, sendo necessário recorrer a técnicas mais profundas para inspeção do binário, tais como a análise estática e a análise dinâmica [Gandotra et al. 2014].

Segundo um relatório publicado pela Malwarebytes [Malwarebytes 2020] em 2020, foi constatado um aumento de 13% nas detecções de *malware* focado no setor

corporativo. O documento também mostra que atividades de *ransomware* estão sendo cada vez mais registradas. O relatório ainda destaca as frequentes infecções por *adwares* em dispositivos móveis. Este cenário exige o desenvolvimento de novos mecanismos de detecção, mitigação e análise dos diferentes tipos de *malware* desenvolvidos e aperfeiçoados dia após dia.

A análise de *malware* pode ser classificada em estática ou dinâmica, onde uma técnica complementa a outra [Chakkaravarthy et al. 2019]. Na análise estática, o usuário realiza a inspeção do binário (cabeçalhos, *strings*, capacidades, etc.), com o objetivo de definir sua função. Em contraste, na análise dinâmica, o usuário executa o binário em um ambiente controlado (máquina virtual ou *sandbox*), com o objetivo de monitorar seu comportamento (chamadas de sistema, consultas DNS, arquivos criados, etc.) [Aslan and Samet 2020].

Com o objetivo de dificultar a inspeção (p. ex. engenharia reversa), atacantes tem se empenhado cada vez mais em aperfeiçoar o uso de técnicas anti-análise. As principais são: (*i*) *anti-disassembly* (para evitar que o código *assembly* seja analisado corretamente); (*ii*) *anti-debugging* (para impedir a execução do programa em *debuggers*); e (*iii*) *anti-VM* (para ocultar o real comportamento do binário em máquinas virtuais) [Singh and Singh 2018].

Embora diversas iniciativas tenham sido desenvolvidas para fornecer suporte à análise de artefatos maliciosos, como o VirusTotal[1] e Any.Run[2], nenhuma delas atualmente oferece uma forma de *deploy* local, na qual o usuário tem o total controle do binário a ser inspecionado. Vale notar que o arquivo (malicioso ou não) é muitas vezes compartilhado com terceiros, trazendo preocupações aos profissionais que precisam manter o sigilo e proteção dos dados analisados. Ademais, tais mecanismos possuem APIs limitadas, exigindo que o usuário adquira licenças de uso para consumir maiores quantidades de dados processados. Desse modo, é necessário o desenvolvimento de soluções capazes de criar melhores ambientes para a análise de *malware*.

Com base no exposto, este trabalho apresenta o `Freki`, uma nova ferramenta para análise automatizada de programas maliciosos. `Freki` é capaz de realizar rapidamente a análise estática de binários, fornecendo um relatório detalhado e compreensível aos usuários, além de permitir a posterior consulta e *download* dos resultados através de uma interface principal ou via API REST fornecida nativamente pelo sistema. Vale notar que a intenção não é substituir as ferramentas atuais, mas fornecer uma solução *open-source*, modular e de fácil implantação local.

O restante deste artigo está estruturado da seguinte forma: a Seção 2 apresenta a solução proposta e detalha sua arquitetura, enfatizando seus principais componentes; a Seção 3 contém as instruções para utilização do `Freki` e a demonstração planejada; a Seção 4 apresenta as considerações finais e delineia apontamentos de trabalhos futuros.

2. Freki: Arquitetura e Implementação

A solução proposta nesse trabalho é uma ferramenta *open-source* capaz de analisar estaticamente arquivos binários, com foco em programas maliciosos. O principal objetivo

[1]https://www.virustotal.com
[2]https://any.run

é auxiliar os usuários na inspeção dos artefatos e averiguação das funcionalidades do arquivo. A análise permite a identificação de diversas características do binário, tais como: *hashes*, avaliação por antivírus, arquitetura, cabeçalhos, seções, importações e arquivos ocultos. Esses resultados podem orientar os usuários nas análises dinâmicas e auxiliar na classificação das diversas famílias de *malware*.

2.1. Arquitetura

A arquitetura do `Freki` é composta por diferentes módulos especializados em uma etapa da análise dos artefatos. `Freki` disponibiliza uma interface única e de fácil utilização para análise de binários, além de permitir a criação de relatórios e a posterior consulta a arquivos já analisados via seu *hash* SHA-1. A Figura 1 apresenta uma visão geral da arquitetura do `Freki` e seus módulos funcionais. Uma das principais vantagens da ferramenta é a sua simplicidade, além da fácil instalação, viabilizada através de recursos de conteinerização via Docker[3].

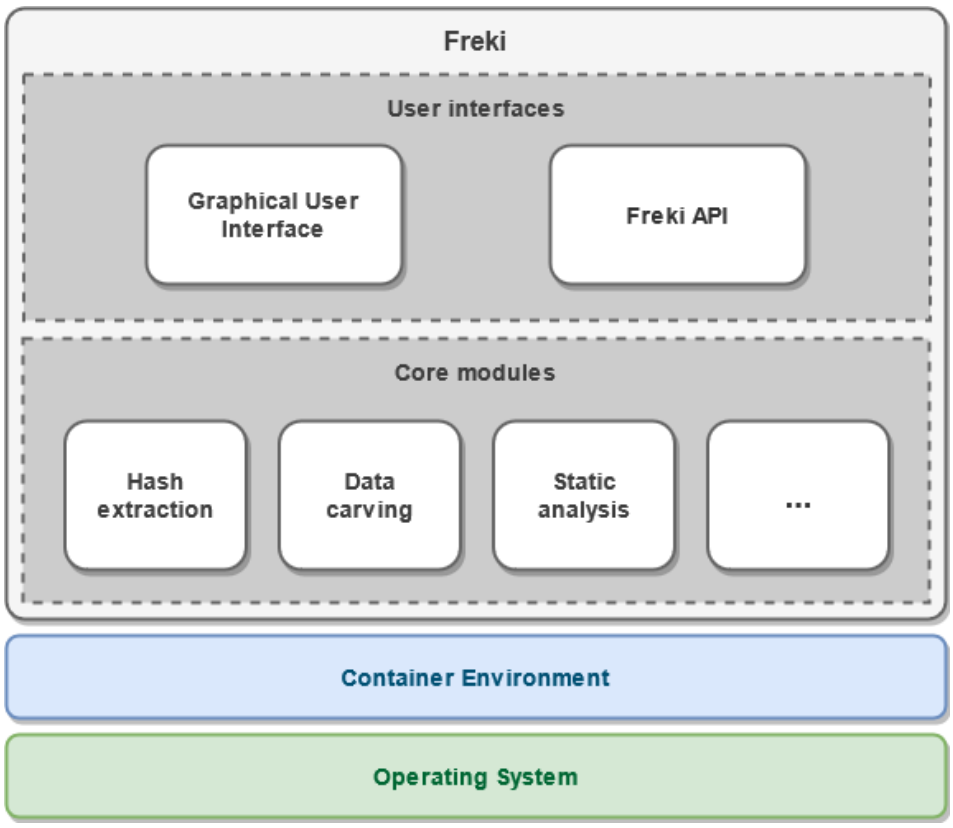

Figura 1. Arquitetura do Freki.

A ferramenta consiste em três componentes principais: (*i*) *user interfaces*, onde é possível utilizar uma plataforma *web* para envio e consulta dos arquivos analisados ou realizar solicitações à sua API REST via linha de comando; (*ii*) *core modules*, local das funcionalidades principais do sistema e onde todas as análises são orquestradas; e (*iii*) *container environment*, que garante o isolamento, fácil instalação e gerenciamento da aplicação.

[3]https://www.docker.com/

Como ilustrado na Figura 2, o ambiente virtualizado através dos *containers* é composto por três elementos: (*i*) servidor *web* Nginx[4] atuando como *proxy* reverso para a aplicação principal; (*ii*) a própria aplicação, que se comunica com os demais *containers* para envio e armazenamento de dados; e (*iii*) instância de um banco de dados relacional (implementado através do MariaDB[5]), responsável por armazenar os resultados das análises e responder as solicitações da aplicação.

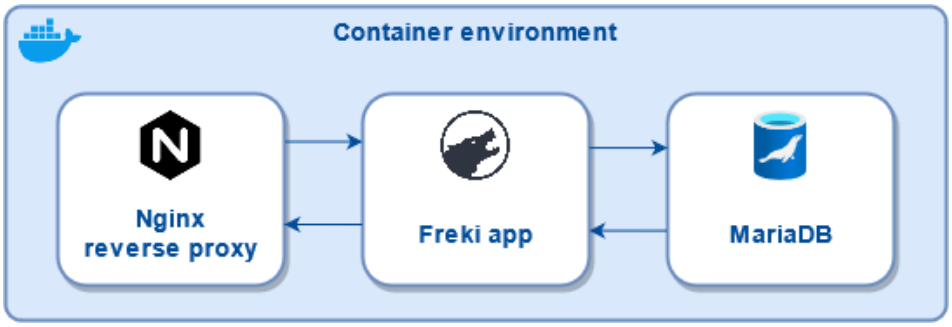

Figura 2. Ambiente de *containers* do Freki.

As subseções a seguir descrevem os principais componentes do `Freki`, seus módulos e funcionalidades. Vale notar que a a arquitetura modular do `Freki` foi planejada para viabilizar a fácil manutenção e inclusão de novos módulos pelos seus usuários, elevando seu potencial de análises e garantindo resultados coerentes para diferentes tipos de arquivos.

2.1.1. User Interface (UI)

`Freki` apresenta uma interface simples e autocontida, na qual os usuários podem facilmente navegar entre os principais recursos disponíveis. A página principal apresenta aos usuários os últimos arquivos enviados para análise, bem como permite o acesso a estes dados para armazenamento local. Após o envio do binário a ser analisado, o usuário é redirecionado para a página de análise exclusiva de cada arquivo, que contém detalhes e características do programa inspecionado - como exposto na Figura 3. Por questões de segurança, é necessário criar uma conta na plataforma para o envio de arquivos. Entretanto, a navegação e descarregamento dos arquivos podem ser realizadas por qualquer usuário. Vale notar que todos os arquivos disponibilizados para *download* são compactados e protegidos por senha. Ademais, a aplicação permite que os usuários autenticados façam comentários, permitindo uma melhor comunicação entre os usuários a respeito dos resultados das análises.

2.1.2. REST API

Visando expandir a usabilidade da ferramenta, uma API REST foi desenvolvida e integrada à aplicação. Desse modo, é possível realizar consultas ao `Freki` através de diferentes ferramentas e linguagens de programação. Logo, novos mecanismos para análise

[4]https://www.nginx.com
[5]https://mariadb.org

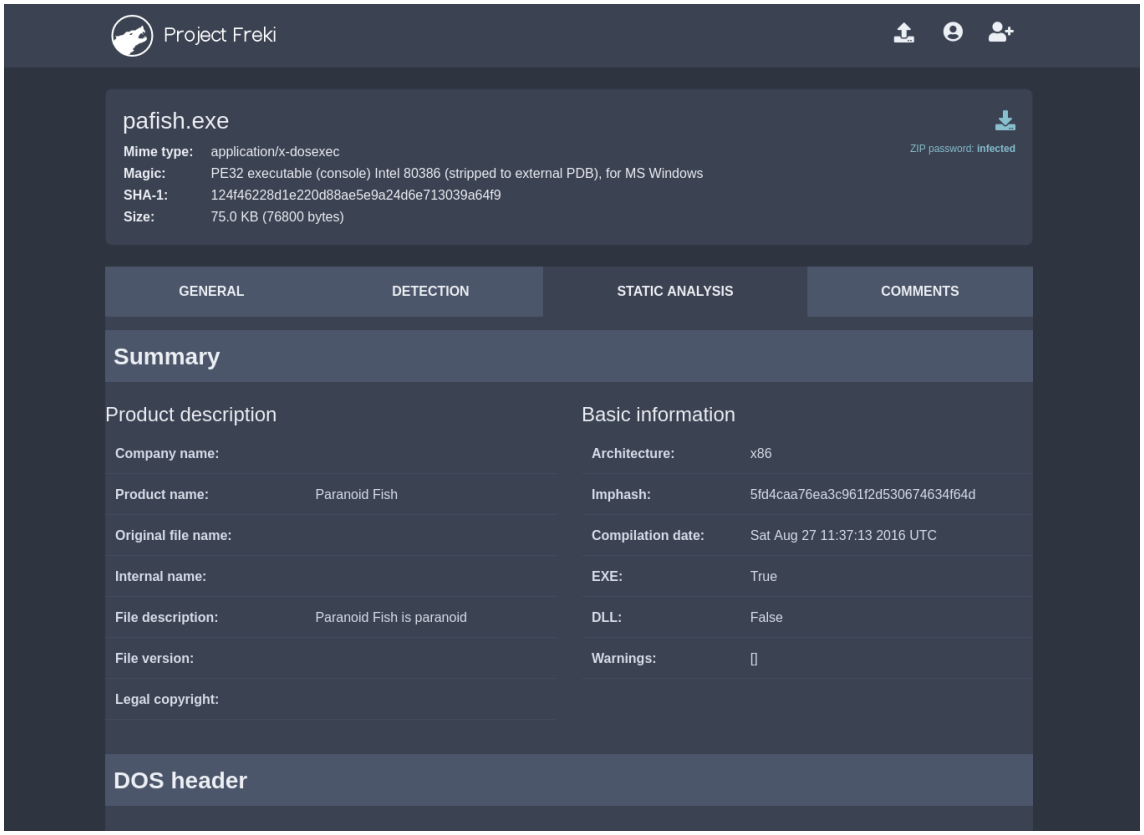

Figura 3. Interface principal do Freki.

de *malware* podem consultar diretamente o `Freki` e utilizar os dados fornecidos internamente para os mais diferentes propósitos, como resposta a incidentes e análise forense. Visando dificultar ataques de negação de serviço, por padrão, há um limite de 4 requisições por segundo à API. Porém, esse valor pode ser customizado pelos proprietários das instâncias do `Freki` dependendo da necessidade. Toda a definição, criação e documentação da API foi realizada através do *framework* Swagger[6].

2.1.3. Core Modules

O *core* do `Freki` é o componente responsável por concentrar as principais funcionalidades utilizadas durante a execução da aplicação. Esse componente contém os módulos que atualmente abrangem as seguintes funções: (*i*) extração dos *hashes* do arquivo; (*ii*) consultas à plataforma VirusTotal; (*iii*) análise estática; (*iv*) *data carving*; e (*v*) classificação do binário. Uma breve descrição de cada funcionalidade é fornecida a seguir:

- ***Extração dos hashes***: são calculados vários *hashes* com o objetivo de facilitar a identificação do binário dentro do `Freki` e na Internet. Até o momento, os seguintes *hashes* são suportados: MD5, SHA-1, SHA-256, SHA-384, SHA-512, CRC32 e SSDEEP;

[6]https://swagger.io

- ***Consultas ao VirusTotal***: a plataforma VirusTotal fornece uma API de fácil utilização para o envio de binários a diferentes antivírus (p. ex. Avast, Avira, ClamAV e Kaspersky). Desse modo, é possível garantir um maior nível de acurácia na identificação de um arquivo malicioso. Caso o usuário forneça a sua chave de API do VirusTotal, o `Freki` é capaz de realizar consultas automáticas a essa API e apresentar os dados fornecidos pelos diferentes motores. Caso contrário, o administrador também pode fornecer uma chave mestra que será utilizada para consultar os arquivos enviados por todos os usuários da instância;

- ***Análise estática***: a análise estática é uma etapa essencial para identificação de várias características do executável. Diante disso, `Freki` é capaz de extrair diversas informações a respeito do arquivo em questão, tais como arquitetura, cabeçalhos, seções, importações, capacidades e *strings*;

- ***Data carving***: muitos programas maliciosos possuem arquivos ocultos que necessitam de técnicas especiais para extração. Logo, `Freki` provê um mecanismo para *data carving*, possibilitando a extração dos artefatos antes da execução do binário;

- ***Classificação do binário***: visando auxiliar usuários e pesquisadores na identificação e classificação dos arquivos analisados, a aplicação é capaz de identificar características e famílias de *malware* através de regras de classificação predefinidas.

2.2. Implementação

O *back-end* do `Freki` é totalmente implementado na linguagem `Python`, fazendo uso das seguintes bibliotecas que automatizam o trabalho de implementação: o *framework web* utilizado foi o `Flask` [Pallets 2010], devido a sua grande modularidade e simplicidade, juntamente com o servidor `Gunicorn` [Chesneau 2009]. Para acesso ao banco, foi utilizado o ORM (*Object Relational Mapper*) `SQLAlchemy` [SQLAlchemy 2018]. O *parsing* das informações contidas em executáveis PE (*Portable Executable*) é realizado através da biblioteca `pefile` [Carrera 2015] e a identificação das suas capacidades é realizada através do `capa` [FireEye 2020]. Já o processo de *data carving* é realizado pelo `Foremost` [Kris Kendall and Mikus 2001]. Por fim, a classificação dos artefatos é realizada através do `Yara` [VirusTotal 2012] e mais um conjunto predefinido de regras fornecidas pela comunidade[7]. O *front-end* da ferramenta foi construído através do *framework* `Bootstrap`[8], garantindo um fácil desenvolvimento de *design* responsivo dos componentes em diferentes dispositivos.

O ambiente base para execução do `Freki` é operacionalizado sob a plataforma Docker. A orquestração do ambiente foi estruturada através de múltiplos *Dockerfiles*, criados para instruir ao Docker Compose[9] os passos necessários para o correto funcionamento da aplicação. Além disso, uma rede privada exclusiva para o `Freki` é criada ao

[7]https://github.com/Yara-Rules/rules
[8]https://getbootstrap.com
[9]https://docs.docker.com/compose

iniciar a aplicação, expondo apenas as portas essenciais para o acesso por parte do usuário e garantindo um maior isolamento do ambiente.

3. Experimentação

Maiores detalhes sobre o `Freki`, como o código-fonte, funcionamento básico e manuais para instalação e utilização (além de um vídeo da ferramenta em funcionamento) podem ser encontrados no repositório oficial do projeto[10]. É possível realizar a instalação da ferramenta de forma nativa, sem a necessidade do Docker. No entanto, o processo de instalação simplificado é descrito a seguir:

1. **Acesso ao código-fonte:** `git clone https://github.com/crhenr/freki.git`;

2. **Alteração do arquivo `.env`:** este arquivo permite a configuração das credenciais desejadas (é recomendado o uso de uma boa política de senhas);

3. **Configuração do HTTPS**: é recomendável habilitar o protocolo HTTPS (principalmente se a instância estiver diretamente exposta à Internet). Para isso, é necessário adicionar o certificado e a chave privada no diretório `docker/nginx/certs`;

4. **Execução do `Freki`**: `make run` ou `docker-compose up -d`.

3.1. Demonstração

O código-fonte, documentação e instruções para instalação via Docker ou nativa estão disponíveis em `https://github.com/crhenr/freki`. O plano para demonstração do `Freki` no salão de ferramentas é apresentar o comportamento da solução diante da análise automatizada de exemplares de *malware* reais, proporcionando aos conferencistas uma visão geral do processo de análise de programas maliciosos e como o `Freki` pode facilitar esse processo. Também será apresentado o processo de utilização da API do `Freki` em diferentes cenários, evidenciando a extensibilidade da ferramenta. Uma demonstração em vídeo da ferramenta pode ser encontrada em: `https://youtu.be/brvNUPgw7ho`.

4. Considerações finais e trabalhos futuros

A análise de *malware* é fundamental para a segurança da informação. Conforme as famílias de programas maliciosos evoluem, novas ferramentas se fazem necessárias para detectar, mitigar e analisar tais ameaças. Neste trabalho propomos o `Freki`, uma ferramenta para análise automatizada de *malware*. A arquitetura da solução fornece um ambiente completo para análise de programas maliciosos de forma simples e intuitiva. Além disso, é possível estender as funcionalidades da aplicação através de customizações no código-fonte ou utilizando a sua API.

Como trabalhos futuros, pretendemos adicionar o suporte a uma quantidade maior de tipos de arquivos (p. ex. `ELF`, `PDF`, `APK`, etc.), visando auxiliar os usuários na

[10]https://github.com/crhenr/freki

identificação de ameaças para diferentes plataformas, incluindo dispositivos habilitados para o cenário da Internet das Coisas (IoT, do inglês *Internet of Things*). Também serão implementados recursos de análise dinâmica dos artefatos em diferentes plataformas através de máquinas virtuais (VMs, do inglês *Virtual Machines*) especialmente projetadas para essa finalidade. Ademais, pretendemos aperfeiçoar ainda mais o código-fonte e a arquitetura do `Freki`, garantindo o maior desempenho possível nas análises.

5. Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e ao Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN) pelo apoio ao desenvolvimento desse trabalho.

Referências

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

Carrera, E. (2015). pefile. `https://github.com/erocarrera/pefile`.

Chakkaravarthy, S. S., Sangeetha, D., and Vaidehi, V. (2019). A survey on malware analysis and mitigation techniques. *Computer Science Review*, 32:1–23.

Chesneau, B. (2009). Gunicorn wsgi http server for unix. `https://gunicorn.org/`.

FireEye (2020). capa. `https://github.com/fireeye/capa`.

Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 2014.

Kaur, G. and Nagpal, B. (2012). Malware analysis & its application to digital forensic. *International Journal on Computer Science and Engineering (IJCSE)*, 4(04):622–626.

Kris Kendall, J. K. and Mikus, N. (2001). Foremost. `http://foremost.sourceforge.net`.

Malwarebytes (2020). 2020 state of malware report. `https://www.malwarebytes.com/resources/files/2020/02/2020_state-of-malware-report-1.pdf`.

Pallets (2010). Flask documentation. `https://flask.palletsprojects.com/en/2.0.x/`.

Singh, J. and Singh, J. (2018). Challenge of malware analysis: malware obfuscation techniques. *International Journal of Information Security Science*, 7(3):100–110.

SQLAlchemy (2018). Sqlalchemy - the database toolkit for python. `https://www.sqlalchemy.org/`.

VirusTotal (2012). Yara. `https://github.com/VirusTotal/yara`.