

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Vieira de Figueiredo

***Sandbox as a Service: automatizando a
configuração do Cuckoo Sandbox e a geração
de dados para análise de malware***

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Vieira de Figueiredo

***Sandbox as a Service: automatizando a configuração do
Cuckoo Sandbox e a geração de dados para análise de
malware***

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Renan Gonçalves Cattelan

Universidade Federal de Uberlândia – UFU
Faculdade de Ciência da Computação
Bacharelado em Ciência da Computação

Uberlândia, Brasil

2022

Guilherme Vieira de Figueiredo

Sandbox as a Service: automatizando a configuração do Cuckoo Sandbox e a geração de dados para análise de malware

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 31 de maio de 2022:

Prof. Dr. Renan Gonçalves Cattelan
Orientador

Professor

Professor

Uberlândia, Brasil
2022

Agradecimentos

Agradeço profundamente à minha mãe Cynthia, meus irmãos Gabriel e Laura que sempre me apoiaram e me encorajaram a seguir meus sonhos. À minha vó Selvita, meu pai Ruy e todos os meus familiares que sempre me ajudaram independente do que acontecesse, o apoio de cada um de vocês foi e continua sendo muito importante para mim.

Agradeço também ao meu orientador Renan, por toda paciência que teve comigo e por toda a motivação e conhecimento que me passou, assim como ao professor Rodrigo, obrigado por terem me mostrado o caminho e me ajudado nesta etapa extremamente importante da minha vida.

Por fim gostaria de agradecer a todos os professores, técnicos e funcionários da UFU, que fazem esse trabalho maravilhoso, possibilitando que diversas pessoas tenham a oportunidade de estudar e aprender. Obrigado!

Resumo

Este trabalho de conclusão de curso busca criar uma plataforma unificada de análise dinâmica e estática de *malware*, na qual diversos pesquisadores possam gerar conjuntos de dados para utilização em suas pesquisas. Propõe-se que a geração de tais conjuntos seja flexível o suficiente para abranger as diversas áreas de pesquisa de *malware*, permitindo escolher entre os campos obtidos a partir da fase de análise, que poderão ser incluídos ou não no conjunto de dados resultante. Através de extensões nas funcionalidades da plataforma *open-source Cuckoo Sandbox*, que anteriormente permitia apenas downloads de *reports* individuais no formato *JSON*, foram implementadas novas funcionalidades, fazendo com que a mesma permita o download em lote de *reports* gerados pela fase de análise estática e dinâmica e já no formato *CSV*, que é um formato mais adequado para algoritmos de aprendizado de máquina.

Palavras-Chave: Análise Estática, Análise Dinâmica, *Cuckoo Sandbox*, *Malware*, *Dataset*, Conjunto de dados, *Open-source*.

Lista de ilustrações

Figura 1	–	Ciclo <i>Request/Response</i> do Django <i>framework</i> .	16
Figura 2	–	Arquitetura do <i>Cuckoo Sandbox</i> .	18
Figura 3	–	Requisição ao <i>endpoint</i> <code>api/v3/files</code> da <i>API</i> do <i>VirusTotal</i> e seu retorno.	19
Figura 4	–	Requisição ao <i>endpoint</i> <code>api/v3/analyses/{id}</code> da <i>API</i> do <i>VirusTotal</i> e seu retorno.	19
Figura 5	–	Página inicial da plataforma <i>Cuckoo Web</i> .	21
Figura 6	–	Página de configuração de opções de análise.	22
Figura 7	–	Página de exibição dos status de arquivos enviados - em processamento.	22
Figura 8	–	Página de exibição dos status de arquivos enviados - concluído.	22
Figura 9	–	Máquina virtual executando arquivo.	23
Figura 10	–	<i>Collections</i> do <i>Schema Cuckoo</i> .	24
Figura 11	–	Estrutura dos dados pertencentes à <i>collection analysis</i> .	24
Figura 12	–	Estrutura hierárquica dos dados do campo <i>info</i> , pertencente à <i>collection analysis</i> .	25
Figura 13	–	Exemplo de como ficam armazenadas as <i>URLs</i> acessadas por um arquivo.	25
Figura 14	–	Item adicionado a <i>Navbar</i> da plataforma <i>Cuckoo Web</i> .	26
Figura 15	–	Linha de código incluída no arquivo <code>urls.py</code> .	26
Figura 16	–	Trecho de código incluído no arquivo <code>analysis/views.py</code> .	26
Figura 17	–	Representação <i>Front-end</i> para campos unários.	27
Figura 18	–	Representação <i>Front-end</i> para campos do tipo lista, sem dados adicionados.	28
Figura 19	–	Representação <i>Front-end</i> para campos do tipo lista, com dados adicionados.	28
Figura 20	–	Código do evento de clique do botão <i>Export</i> .	29
Figura 21	–	Código da função <code>get_tree_selected_names()</code> .	29
Figura 22	–	Código da função <code>get_adds()</code> .	30
Figura 23	–	Exemplo de retorno da função <code>get_tree_selected_names()</code> .	30
Figura 24	–	Exemplo de retorno da função <code>get_adds()</code> .	30
Figura 25	–	Requisição <i>POST</i> feita à <i>API</i> do <i>Cuckoo Sandbox</i> .	31
Figura 26	–	Código para buscar campos da <i>collection analysis</i> .	31
Figura 27	–	Código para extração dos valores dos campos unários selecionados pelo usuário.	32
Figura 28	–	Código para extração dos valores dos campos do tipo lista adicionados pelo usuário.	33
Figura 29	–	Resposta a requisição feita a <i>API</i> do <i>Cuckoo Web</i> .	33
Figura 30	–	Código para download do arquivo <i>CSV</i> gerado.	34

Figura 31 – Exemplo de conjunto de dados, gerado no formato *CSV*. 34

Lista de abreviaturas e siglas

JSON	JavaScript Object Notation
CSV	Comma-Separated Value
BSON	Binary JSON
API	Application Programming Interface
VM	Virtual Machine
IP	Internet Protocol
HTML	HyperText Markup Language
CSS	Cascading Styles Sheets
MVC	Model View Controller
URL	Uniform Resource Locator
DLL	Dynamic-Link Library
AJAX	Asynchronous JavaScript and XML
BLOB	Binary Large Object

Sumário

1	INTRODUÇÃO	9
1.1	Contextualização	9
1.2	Motivação	9
1.3	Objetivos	10
1.4	Organização do Texto	10
2	REVISÃO BIBLIOGRÁFICA	11
2.1	Fundamentação Teórica	11
2.2	Trabalhos Relacionados	13
3	TECNOLOGIAS	15
3.1	Front-End	15
3.1.1	HTML	15
3.1.2	CSS	15
3.1.3	Bootstrap	15
3.1.4	jQuery	15
3.2	Back-End	16
3.2.1	Python	16
3.2.2	Django	16
3.2.3	MongoDB	17
3.2.4	Cuckoo Sandbox	17
3.2.5	VirusTotal API	19
4	DESENVOLVIMENTO	20
4.1	Análise de novos arquivos	21
4.2	Estrutura da Base de Dados	24
4.3	Geração do conjunto de dados	26
4.3.1	Implementação Front-End	27
4.3.2	Implementação Back-End	31
4.3.3	Download do conjunto de dados	34
5	CONCLUSÕES	35
	REFERÊNCIAS	36

1 Introdução

1.1 Contextualização

Malwares, ou códigos maliciosos, são programas criados e projetados unicamente para causar danos e executar atividades maliciosas em um computador (CERT.BR, 2017). Assim, para melhor compreender e permitir combater esse tipo de ameaça, surge a área de análise de *malware*.

A análise de *malware* é uma área que estuda como códigos maliciosos funcionam, aplicando diversas técnicas para obter tal resultado. As duas principais técnicas que são utilizadas para realização de tais análises, são:

- Análise estática: Método de análise que é feita sem a execução do *malware*. É onde ocorre a verificação por antivírus, verificação de *hash* e etc. (YUSIRWAN; PRAYUDI; RIADI, 2015).
- Análise dinâmica: Método de análise que é feita executando o *malware*, em uma máquina virtual, para tornar a análise mais segura. É onde ocorre a monitoração dos pacotes enviados, registros modificados e etc. (YUSIRWAN; PRAYUDI; RIADI, 2015).

Nesse contexto, tem-se também os antivírus, que são *softwares* criados com o intuito de detectar *malwares* no momento que eles tentam infectar um sistema. Basicamente, eles utilizam um mecanismo de assinaturas, pelo qual conseguem verificar em um banco de dados se tal arquivo é ou não um código malicioso. O problema com esse método é que os antivírus só conseguem detectar *malwares* que já são conhecidos por esses bancos de dados. Então, mesmo que o antivírus esteja instalado, é possível que o sistema seja infectado por um vírus novo, por isso os mesmos sofrem constantes atualizações, adicionando novas ameaças descobertas ao seu banco de dados.

1.2 Motivação

Como os antivírus atuais, por conta da natureza de sua implementação, possuem problemas para lidar com códigos maliciosos novos, os *malwares* continuam causando diversos prejuízos e problemas. Dessa forma, se torna de extremo interesse obter uma ferramenta que permita detectar *malwares* antes dos mesmos serem conhecidos. Assim, posteriormente utilizando-se de técnicas de aprendizado de máquina, que já tiveram su-

cesso em diversas áreas (análise de imagens, detecção de spam e etc)(SHAKYA, 2020), (DOUZI et al., 2020), seria possível atingir tal objetivo.

Porém, para que seja possível a utilização de tais técnicas, se faz necessário o uso de conjuntos de dados (*datasets*), sendo estes criados a partir dos dados obtidos da análise estática e dinâmica. Atualmente, os pesquisadores têm gasto uma quantidade considerável de tempo na criação dos mesmos, tempo esse que poderia estar sendo melhor aproveitado no estudo das técnicas de aprendizado de máquina, análise dos componentes principais e etc.

1.3 Objetivos

O objetivo deste trabalho é desenvolver/construir uma plataforma que padronize o ambiente de análise e o carregamento dos conjuntos de dados como um serviço computacional (*Sandbox as a Service*) - que irá executar como uma máquina virtual em algum domínio, oferecendo seus serviços para que não seja necessário usar a própria máquina para realizar a análise. Para que isso ocorra, será feito o uso da plataforma *Cuckoo Sandbox* sobre a qual serão realizadas configurações, assim como extensões em suas funcionalidades, de modo que esta passe a disponibilizar uma seção, onde seja possível gerar conjuntos de dados personalizados, contendo dados obtidos a partir da análise estática e dinâmica de todos os arquivos que já foram analisados por ela, assim abrindo uma gama de novas possibilidades para o grupo de pesquisa.

1.4 Organização do Texto

O restante do texto encontra-se organizado da seguinte forma: o Capítulo 2 traz a fundamentação e alguns trabalhos relacionados à proposta; o Capítulo 3 elenca as principais tecnologias empregadas; o Capítulo 4 detalha o desenvolvimento do serviço computacional criado; e, por fim, o Capítulo 5 apresenta as conclusões e considerações finais.

2 Revisão Bibliográfica

2.1 Fundamentação Teórica

Análise de *malware* é o processo onde se busca descobrir o seu comportamento e sua estrutura, assim entendendo melhor como tal código malicioso funciona. A análise de *malware* normalmente é dividida em 2 partes principais: análise estática, cujo método de análise é realizado a partir do arquivo executável do *malware* sem executá-lo; e dinâmica, que é o método de análise onde o *malware* é executado em um sistema seguro. Além disso, tais partes podem ser divididas em sub-conjuntos (YUSIRWAN; PRAYUDI; RIADI, 2015) onde temos:

- **Análise Estática Básica:** No método básico de análise estática, o código que supostamente é um *malware* é testado utilizando um antivírus, fazendo *hashing*, e detecção de ofuscação.
- **Análise Estática Avançada:** No método avançado de análise estática, é feita uma análise mais profunda, realizando análise de *strings*, bibliotecas e funções, além da utilização de um *disassembler*.
- **Análise Dinâmica Básica:** No método básico de análise dinâmica, é criada uma máquina virtual, que será utilizada para a realização da análise, tal análise é feita através do monitoramento de processos e análise de pacotes.
- **Análise Dinâmica Avançada:** No método de análise dinâmica avançada, é feita uma análise mais profunda, realizando *debugging* e análise de registros.

Para a realização da análise, são necessárias diversas ferramentas, entre elas estão:

- **Cuckoo Sandbox:** Um sistema automatizado para análise dinâmica de *malwares*.
- **VirusShare:** Base de dados contendo diversos *malwares* disponíveis para download, para pesquisa.
- **VirusTotal:** Outra base de dados contendo diversos *malwares*, além da implementação de uma *API* para análise estática.

Tais ferramentas serão discutidas com mais detalhes no Capítulo 3, uma vez que elas são de extrema importância para este trabalho.

Após a realização dessas etapas de análise, é gerado um relatório contendo os dados obtidos em cada uma das etapas e, através desses dados, é possível entender melhor o funcionamento do código malicioso, assim como construir *datasets*, que poderão, posteriormente, ser utilizados como entrada em modelos de aprendizado de máquina que tenham como objetivo detectar *malwares* de acordo com seu comportamento no sistema.

Estes modelos de aprendizado de máquina, que se baseiam no comportamento do *malware*, têm sido alvo de diversas pesquisas, uma vez que os antivírus atuais que utilizam detecção baseada em assinaturas, pela natureza de sua implementação, possuem dificuldades ao lidar com *malwares* mais novos, pois estes atingiram um nível de maturidade e complexidade, que permite com que eles troquem as assinaturas de seus códigos constantemente (RAD; MASROM; IBRAHIM, 2012). Apesar disso, os códigos maliciosos foram desenvolvidos para realizar uma atividade maliciosa específica. Sendo assim, o seu comportamento, apesar de suas mudanças estruturais, irá permanecer o mesmo (CATAK et al., 2020), o que faz com que dados sobre o comportamento do malware no sistema sejam valiosos.

Dessa forma, tem-se uma eterna luta entre os desenvolvedores dos códigos maliciosos e os profissionais de segurança de informação, onde os desenvolvedores estão sempre tentando encontrar maneiras de esconder a natureza real de seu código, através de técnicas de evasão, enquanto que os profissionais de segurança da informação criam métodos para que isso não ocorra.

Os métodos de evasão podem ser divididos entre aqueles que tentam burlar a análise estática, e aqueles que tentam burlar a análise dinâmica, porém, na prática, tais métodos são usados em conjunto, criando assim uma técnica de evasão híbrida, na tentativa de obter um maior sucesso. Dentre os métodos de evasão de análise estática alguns dos mais populares são:

- **Ofuscação:** Utiliza-se de métodos para tentar impedir a análise estática de encontrar o código malicioso, como por exemplo inverter o código lexicalmente.
- **Esteganografia:** oculta-se o código em uma imagem, por exemplo, para que o *malware* passe despercebido pela análise e seja executado posteriormente (EHTESHAMIFAR et al., 2019).

Dentre os métodos de evasão de análise dinâmica alguns dos mais populares são:

- **Detecção por acesso a Web:** Como quase todos os *sandboxes*¹ desabilitam o tráfego de rede, o *malware* pode tentar acessar serviços Web globais para obter alguma

¹ Um *sandbox* é um ambiente virtual mais seguro que uma *VM*, de modo que tudo que é executado “dentro” dele, não modifica o sistema real (arquivos, registros, *logs* e etc), assim, um dos seus principais

informação que é difícil de ser emulada em um ambiente virtual, como por exemplo serviços de resolução de IP, como *ip-adrr.es* (CHAILYTKO; SKURATOVICH, 2016).

- **Detecção por UI:** Como a análise dinâmica é normalmente automatizada, o *malware* pode tentar verificar se está em um ambiente virtual, verificando o uso de alguns recursos, como por exemplo, o movimento do mouse, o *scroll*, a velocidade de digitação.
- **Detecção por tempo:** Por causa do ambiente virtual usado pelos *sandboxes*, algumas operações possuem uma performance pior do que se estivesse em um ambiente normal, assim o *malware* pode perceber essa demora, e tentar esconder sua natureza (EHTESHAMIFAR et al., 2019).

Para maiores informações sobre tais métodos e algumas possíveis soluções, consulte (EHTESHAMIFAR et al., 2019) e (CHAILYTKO; SKURATOVICH, 2016).

Como se pode ver, existem diversas técnicas de evasão de análise de *malware*, e a maioria dos analisadores de *malware* são vulneráveis a técnicas de evasão. Dentre os 41 analisadores testados, entre eles *AVG*, *Avast* e etc, apenas 2 obtiveram boas taxas de detecção de evasão, ainda mais quando as técnicas são utilizadas em conjunto (EHTESHAMIFAR et al., 2019), o que faz extremamente necessário o estudo e implementação de métodos que identifiquem a evasão. Porém é um pouco mais difícil do que se parece resolver esse problema, uma vez que a implementação de todas as soluções poderia causar um *overhead* computacional, o qual também poderia ser detectado pelo *malware* (FERRAND, 2015).

2.2 Trabalhos Relacionados

Yusirwan, Prayudi e Riadi (2015) utilizaram a análise estática e dinâmica para entender melhor o funcionamento do *malware TT.exe*, que era indetectável por antivírus na época, explicando a cada passo, as ferramentas utilizadas e os resultados obtidos. Tais resultados demonstram a importância da realização de ambas as fases de análise, uma vez que dessa forma é possível obter uma visualização completa do “cenário” do arquivo analisado.

Chailytko e Skuratovich (2016) descrevem os diversos métodos de evasão de *sandboxes* e propõe soluções para os mesmos, incluindo um repositório *GitHub* que contém as implementações das soluções descritas pelos autores. As soluções visam principalmente a

usos é exatamente na execução de códigos conhecidamente maliciosos, ou com o potencial de ser, para que o mesmo não danifique o sistema real.

plataforma do *Cuckoo Sandbox*, que é uma plataforma extensamente utilizada por profissionais da área de segurança de informação e, exatamente por isso, é alvo de uma parte dos métodos de evasão de *sandboxes* existentes, que utilizam detalhes específicos da implementação do *Cuckoo* para detectar o ambiente virtual.

Ehteshamifar et al. (2019) testam diversos analisadores de *malware*, para observar quais tem as melhores defesas contra técnicas de evasão. Tal teste é realizado a partir da criação de PDFs contendo códigos maliciosos, sobre os quais são aplicadas diferentes técnicas de evasão. Posteriormente, estes PDFs são enviados para análise e os resultados obtidos pelos analisadores são debatidos, observando-se assim quais técnicas obtiveram os melhores resultados, seja separadas ou em conjunto.

Guibernau (2020) apresenta alguns métodos de evasão e mostra na prática como um *malware* implementa essas técnicas. Após isso, utilizando o *framework MITRE ATT&CK*, o autor oferece possíveis soluções.

Ferrand (2015) demonstra algumas técnicas de detecção de *sandbox*, mais especificamente voltadas para o *Cuckoo Sandbox*, contendo diversos códigos sobre a implementação das mesmas e também possíveis soluções para tais técnicas.

Catak et al. (2020) utilizam a técnica de aprendizado de máquina *LSTM (Long Short-Term Memory)*, que é bastante utilizada em classificação de textos, sobre um conjunto de dados criados por eles, que contém principalmente informações sobre chamadas a *APIs* do Windows. O trabalho tenta classificar entre 8 tipos diferentes de *malwares* (*Adwares, Trojans, Spywares e etc*) e obtém uma boa acurácia de classificação em todos eles.

Miller et al. (2017) discorrem sobre os conhecimentos ganhos ao construir uma plataforma de análise dinâmica de larga escala, utilizando como base o *Cuckoo Sandbox*, como quais ferramentas de virtualização e configurações obtiveram maior sucesso ao realizar as análises, assim como algumas escolhas de implementação.

Rad, Masrom e Ibrahim (2012) discorrem sobre a história dos *malwares* e sua evolução, principalmente na questão de camuflagem, demonstrando como eles a realizam e os problemas que tais evoluções posam aos antivírus atuais, sendo um dos principais problemas, o surgimento dos *malwares* metamórficos, que são *malwares* que conseguem modificar sua assinatura constantemente, assim dificultando bastante a “vida” dos antivírus.

3 Tecnologias

Para implementar a interface, assim como a parte de processamento por trás da geração do conjunto de dados, foram utilizadas diversas ferramentas, estas serão descritas neste Capítulo.

3.1 Front-End

As tecnologias utilizadas para a criação da interface com o usuário, de modo a deixá-la iterativa e atrativa, assim como facilitando o seu uso foram:

3.1.1 HTML

HTML é uma linguagem de marcação de texto, basicamente ela estrutura o conteúdo de uma página, utilizando *tags*, assim dividindo seus elementos entre títulos, parágrafos, imagens, formulários, links, tabelas, etc. Dessa forma o HTML é utilizado para criar o “esqueleto” de uma página Web. Para mais informações consulte <<https://developer.mozilla.org/pt-BR/docs/Web/HTML>>

3.1.2 CSS

CSS é uma linguagem de folhas de estilo, sua função é personalizar/estilizar documentos como HTML, XHTML, através de blocos de códigos, que descrevem como deve ser o layout de certos elementos, classes e etc. Sendo assim utilizado para tornar a interface mais atrativa para o usuário. Para mais informações consulte <<https://developer.mozilla.org/pt-BR/docs/Web/CSS>>

3.1.3 Bootstrap

Bootstrap é um *framework* da linguagem CSS, que permite criar o estilo da página com um design responsivo e elegante, de maneira simples e rápida. Exatamente por conter elementos de design já prontos para uso, como barras de navegação, acordeons e etc. O seu uso acelera bastante o tempo de desenvolvimento da interface. A página oficial do projeto pode ser encontrado em <<https://getbootstrap.com/>>

3.1.4 jQuery

jQuery é uma biblioteca Javascript, que contém diversas funções que facilitam a criação de páginas dinâmicas, através dele é possível criar *scripts* simplificados, que

possibilitam manipular a árvore DOM ¹, consumir *APIs*, criar eventos e etc. A página oficial do projeto pode ser encontrado em <<https://jquery.com/>>

3.2 Back-End

Nesta seção serão descritas as tecnologias utilizadas para processar e armazenar os dados requisitados pelo *Front-end*

3.2.1 Python

Python é uma linguagem de programação de alto nível, multi-paradigma, multi-plataforma que permite a escrita de programas de maneira rápida. Muito utilizada por possuir diversas bibliotecas e uma sintaxe simples ². A documentação oficial da linguagem pode ser encontrada em <<https://www.python.org/>>

3.2.2 Django

Django é um *framework* para desenvolvimento de aplicativos Web, de código aberto, escrito em Python. Utiliza a arquitetura *Model View Template*, que é bem semelhante a arquitetura *MVC*, utilizada por outros *frameworks* como o .NET.

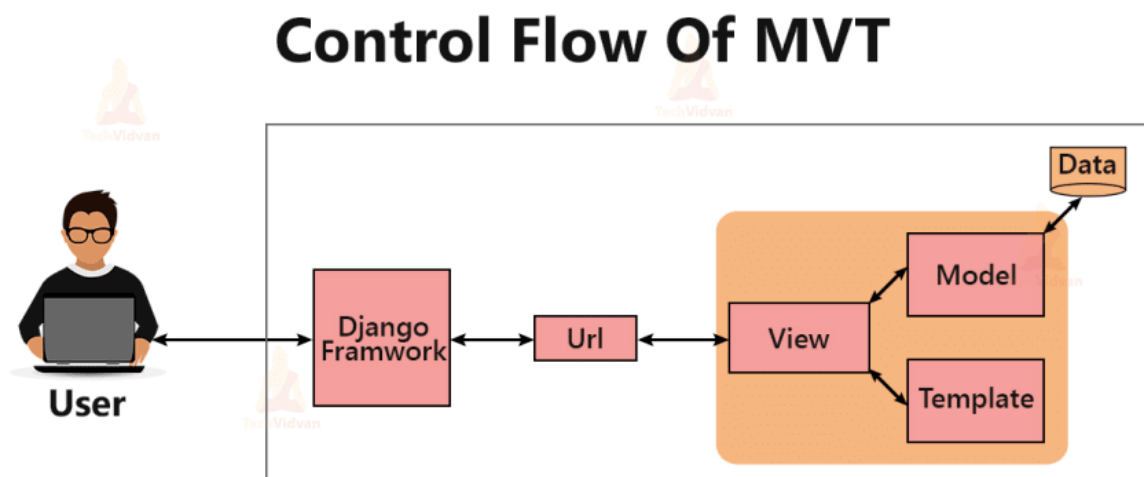


Figura 1 – Ciclo *Request/Response* do Django *framework*.

Fonte: <<https://dev.to/priyanshupanwar/everything-about-django-2-architecture-of-django-206c>>

¹ Estrutura hierárquica do HTML

² Em especial duas dessas bibliotecas foram de extrema importância para este projeto, sendo elas o *pandas*, que foi utilizado na geração do arquivo *CSV* final e o *pymongo* utilizado para buscar os dados salvos no MongoDB.

Ele funciona basicamente como uma estrutura de *Request/Response*, como mostrado na Figura 1, onde a requisição, ao chegar no servidor, é analisada pelo arquivo *urls.py*, que será responsável por direcionar a requisição à *view* correta. A *view* funciona como um *controller* da arquitetura *MVC*, basicamente, ela faz o intermédio entre a camada responsável por gerenciar os dados (*models*) e a camada de interface (*templates*), sua função é então “pedir” os dados para os modelos necessários e então repassá-los ao *template* correto, que será então devolvido como resposta à requisição. A página oficial do projeto pode ser encontrada em <<https://www.djangoproject.com/>>

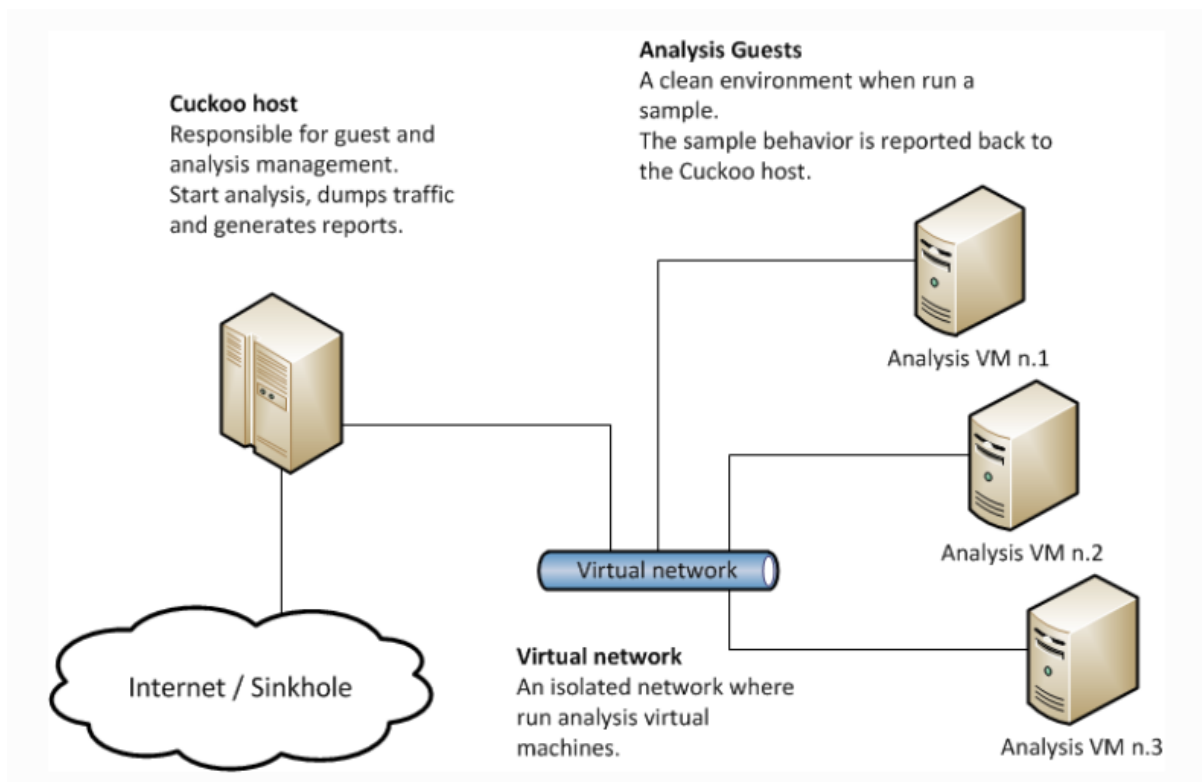
3.2.3 MongoDB

Banco de dados *NoSQL*, bastante flexível, orientado a documentos, onde os dados são armazenados na forma de chave-valor, utilizando arquivos no formato *BSON* (Binary JSON), que é uma extensão do famoso formato *JSON*. Muito utilizado em soluções que lidam com um grande volume de dados. Para mais informações consulte <<https://www.mongodb.com/>>

3.2.4 Cuckoo Sandbox

O *Cuckoo Sandbox* é uma ferramenta de análise dinâmica automatizada, *open-source*, escrita em Python durante o Google Summer Code em 2010. Basicamente, ela é composta por diversos módulos, o que permite extensões em seu código, assim como integrações com outros *softwares*, exatamente por conta de sua estrutura extremamente modular.

Sua arquitetura consiste basicamente de um *host*, de preferência *Linux* x86 bits, que contém o “núcleo” do *Cuckoo*. É ele que é responsável por receber as requisições, gerenciar o processamento de análises e etc, e de diversos *guests*, que são ambientes isolados onde os arquivos enviados pelos usuários serão executados e analisados (Figura 2). Os usuários são capazes de analisar seus arquivos simultaneamente, dependendo da capacidade de processamento do servidor utilizado.

Figura 2 – Arquitetura do *Cuckoo Sandbox*.

Fonte: <https://cuckoo.readthedocs.io/en/latest/introduction/what/>

O envio de arquivos para análise pode ser feito tanto por linha de comando, ou pela interface Web que o *Cuckoo Sandbox* disponibiliza para simplificar o uso. Esse trabalho foi implementado focando apenas na interface Web, para que futuros usuários possam utilizar a plataforma facilmente. A interface Web foi escrita utilizando o *framework* Django como *Back-end*, e no *Front-end* foi utilizado HTML, CSS, Javascript. Já os dados da análise são salvos como documentos no MongoDB, como será mostrado no próximo Capítulo.

Além disso, durante a fase de análise, é possível utilizar diversas ferramentas externas, dependendo daquilo que se deseja alcançar. Uma das ferramentas que foi habilitada neste trabalho foi a *API* do *VirusTotal*, para que além dos dados de análise dinâmica gerados pelo *Cuckoo Sandbox*, seja possível incluir no *dataset* final, também dados de análise estática. Porém, como veremos no próximo Capítulo, o *Cuckoo Sandbox* não disponibiliza nenhuma maneira para download em lote de tais dados, outro problema que será abordado é que o formato destes dados não são muito “amigáveis” à algoritmos de aprendizado de máquina.

3.2.5 VirusTotal API

Uma *API REST* disponibilizada pela equipe do *VirusTotal*, no domínio <<https://www.virustotal.com/>>, que contém diversos *endpoints* que tornam possível a realização de análises estáticas de arquivos através de chamadas a *API*.

Basicamente, os *endpoints* utilizados pelo *Cuckoo Sandbox*, para obter tais dados são:

- **api/v3/files:** *Endpoint* por onde são enviados os binários de um arquivo para o servidor do *VirusTotal*. A resposta dessa requisição será um arquivo *JSON* contendo principalmente o id, através do qual será possível resgatar a análise realizada sobre o arquivo enviado (Figura 3).
- **api/v3/analyses/id:** *Endpoint* no qual através de um id, é possível recuperar os dados da análise estática representados por ele (Figura 4).

```
[5] headers = {
    'x-apikey': '4859f03a03c6dd413fc0a7f92906d051e22c42733e219aad2664aeb219a4f625'
}

data = {
    "file": open('file_teste.png', 'rb')
}

api_url = "https://www.virustotal.com/api/v3/files"

response = requests.post(url = api_url, headers = headers, files = data)
response_files = response.json()
print(response_files)

{'data': {'type': 'analysis', 'id': 'YTQzZWZmZWJ0TlI3MDFmNjNhNzg2NmFkNTM1N2I5NDAGMTY0NzkwNzg0Q=='}}
```

Figura 3 – Requisição ao *endpoint* *api/v3/files* da *API* do *VirusTotal* e seu retorno.

```
[9] id = response_files['data']['id']

api_url = "https://www.virustotal.com/api/v3/analyses/" + id

response = requests.get(url = api_url, headers = headers)
response_analyses = response.json()
print(response_analyses['data']['attributes'])

{'date': 1647907839, 'status': 'completed', 'stats': {'harmless': 0, 'type-unsupported': 14, 'suspicious': 0, 'confirmed-timeout': 0, 'timeou
```

Figura 4 – Requisição ao *endpoint* *api/v3/analyses/{id}* da *API* do *VirusTotal* e seu retorno.

4 Desenvolvimento

Atualmente, a plataforma Web do *Cuckoo Sandbox* não disponibiliza nenhuma maneira para realizar download em lote dos dados¹ gerados pelas análises. Os pesquisadores interessados em tais dados acabam tendo que recorrer ao acesso direto no MongoDB (o que nem sempre é uma opção) e, posteriormente, processar tais dados para fazer a montagem de seu conjunto de dados. Porém, como será visto neste Capítulo, isso não é algo trivial e demanda tempo.

Além disso, até o momento, cada pesquisador precisa fazer a instalação do *Cuckoo Sandbox* em sua máquina, buscar arquivos maliciosos para serem analisados, para só então realizar a análise de fato, na qual os dados resultantes ainda precisarão ser transformados para um formato que seja aceito por modelos de aprendizado de máquina, o que gera um retrabalho que poderia ser evitado com a criação de uma plataforma unificada.

Para evitar esse retrabalho desnecessário e fazendo uso do código já existente do projeto *open-source Cuckoo Sandbox*, disponível em <<https://github.com/cuckoosandbox/cuckoo>>, e das tecnologias citadas no Capítulo anterior, foram implementadas novas funcionalidades na plataforma *Cuckoo Web*, de modo que seja possível gerar os conjuntos de dados desejados (já no formato *CSV*, apropriado para manipulação por ferramentas de aprendizado de máquina) através dela. O código modificado foi disponibilizado em <https://github.com/GuilhermeVF/Cuckoo_Modified_DatasetGenerator>.

Como os pesquisadores, dependendo de sua linha de pesquisa, possuem diferentes requisitos e necessidades para seu conjunto de dados², este trabalho foi pensado de forma a tornar a geração desse conjunto de dados a mais flexível possível, fazendo com que se possa escolher quais informações deverão ser incluídas e quais devem ficar de fora, permitindo dessa forma, também, que sejam testados diferentes *datasets* nos algoritmos de aprendizado, sem muitas dificuldades para a geração dos mesmos.

Os passos a serem seguidos para inclusão de novas análises no conjunto de dados e geração do *dataset*, serão descritos no decorrer deste Capítulo, assim como o que foi necessário para a implementação dessas novas funcionalidades.

¹ É possível fazer downloads de *reports* de arquivos específicos, no formato *JSON*, o que não é ideal para modelos de aprendizado de máquina.

² Um pesquisador estudando *ransomwares*, por exemplo, necessita de informações sobre chamadas de *APIs* de criptografia, enquanto outro que estuda *trojans* não teria tanto interesse nesses dados.

4.1 Análise de novos arquivos

Para realizar a análise de novos arquivos, primeiramente o usuário deverá entrar na plataforma Web, no domínio em que a mesma estiver disponibilizada, onde será direcionado a página inicial (Figura 5), que contém diversos dados, como análises recentes, poder de processamento disponível, entre outras informações.

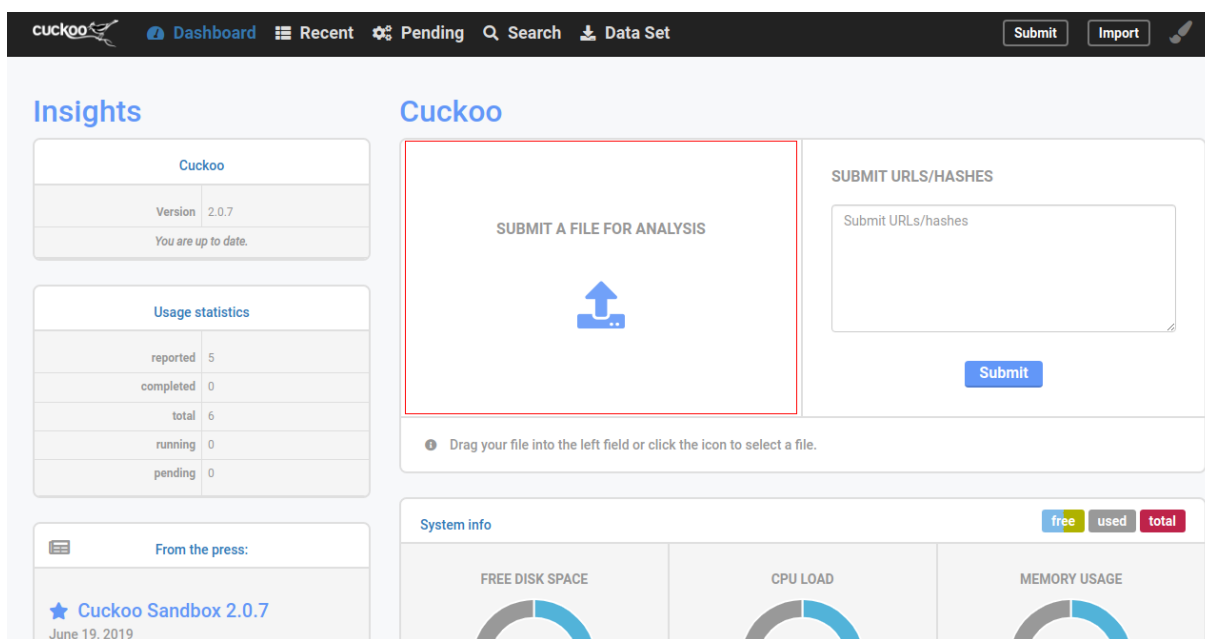


Figura 5 – Pagina inicial da plataforma Cuckoo Web.

Clicando em “SUBMIT A FILE FOR ANALYSIS”, sublinhado em vermelho (Figura 5), será possível selecionar os arquivos que serão analisados. Uma vez selecionados, os arquivos serão mostrados na tela (Figura 6), assim como algumas opções da fase de análise, como *timeout*, tipo de roteamento de rede, etc.

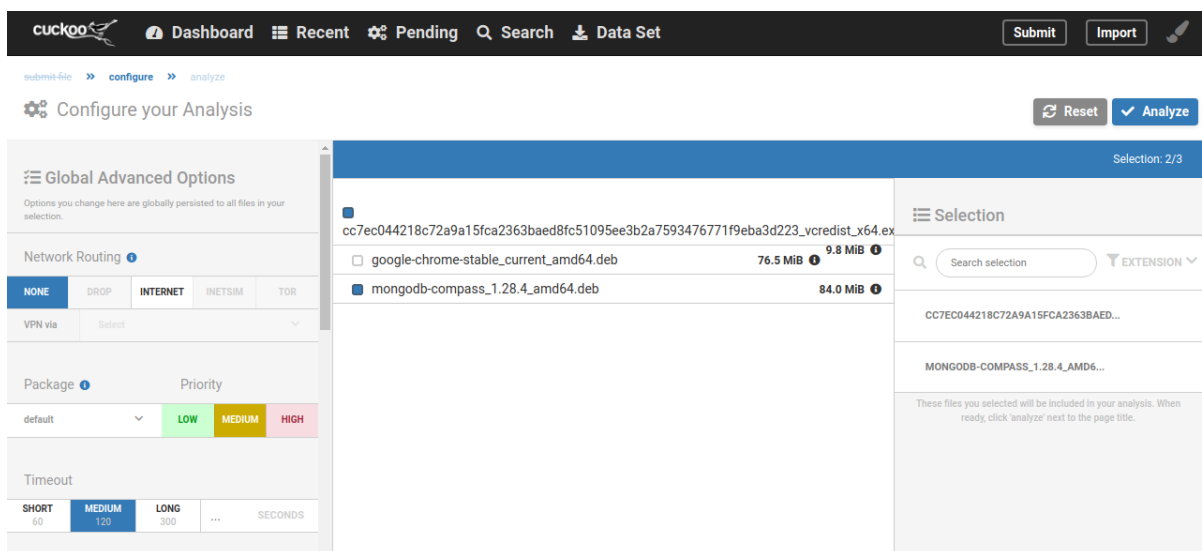


Figura 6 – Página de configuração de opções de análise.

Terminando a configuração das opções desejadas, basta clicar no botão “Analyze”, que será enviada uma requisição ao servidor para que a análise dos arquivos comece de fato. Enquanto isso, o usuário será redirecionado para uma página, onde ele poderá ver o status do processamento de sua requisição (Figura 7 e Figura 8).

Tasks: Refreshes every 2.5 seconds

Task ID	Date	Filename / URL	Package	Status
7	19/03/2022 19:56	cc7ec044218c72a9a15fca2363baed8fc51095ee3b2a7593476771f9eba3d223_vcredist_x64.exe	exe	running
8	19/03/2022 19:56	mongodb-compass_1.28.4_amd64.deb	-	running
Done				

Figura 7 – Página de exibição dos status de arquivos enviados - em processamento.

Tasks: Refreshes every 2.5 seconds

Task ID	Date	Filename / URL	Package	Status
7	19/03/2022 19:56	cc7ec044218c72a9a15fca2363baed8fc51095ee3b2a7593476771f9eba3d223_vcredist_x64.exe	exe	reported
8	19/03/2022 19:56	mongodb-compass_1.28.4_amd64.deb	-	reported
Done				

Figura 8 – Página de exibição dos status de arquivos enviados - concluído.

Uma vez que a requisição chegue ao servidor, serão iniciadas máquinas virtuais (Figura 9), que serão responsáveis pela execução dos arquivos enviados. Essas máquinas

contêm o processo *cuckoo.exe*, que basicamente é encarregado de registrar todas as modificações que a execução de cada arquivo gera no sistema, bem como acessos a *URLs* externas, chamadas de *APIs*, etc. Quando a execução do arquivo for concluída, a máquina virtual responsável pela execução do mesmo voltará ao seu estado original, utilizando *snapshots* do sistema salvos antes do início do processamento.

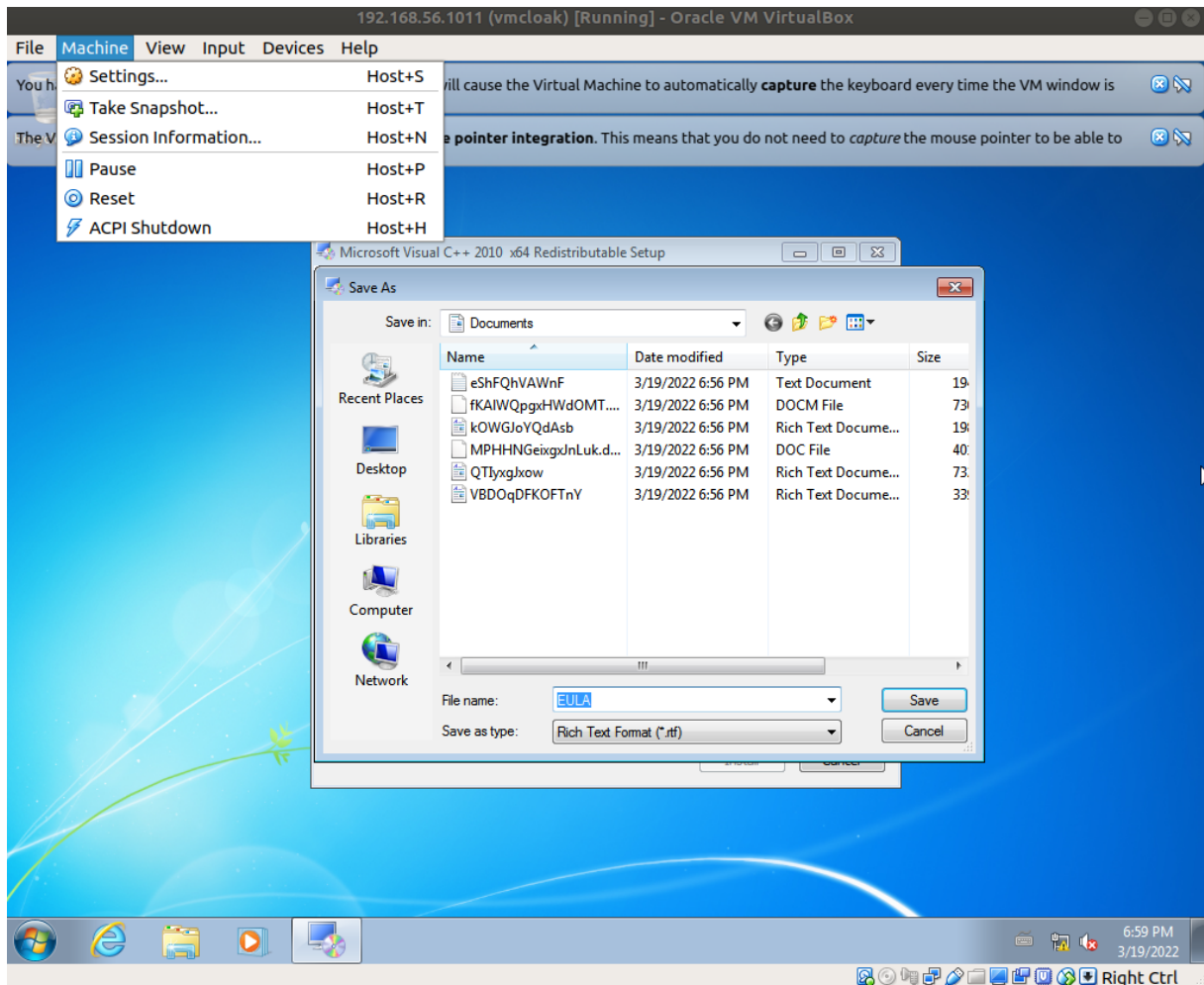
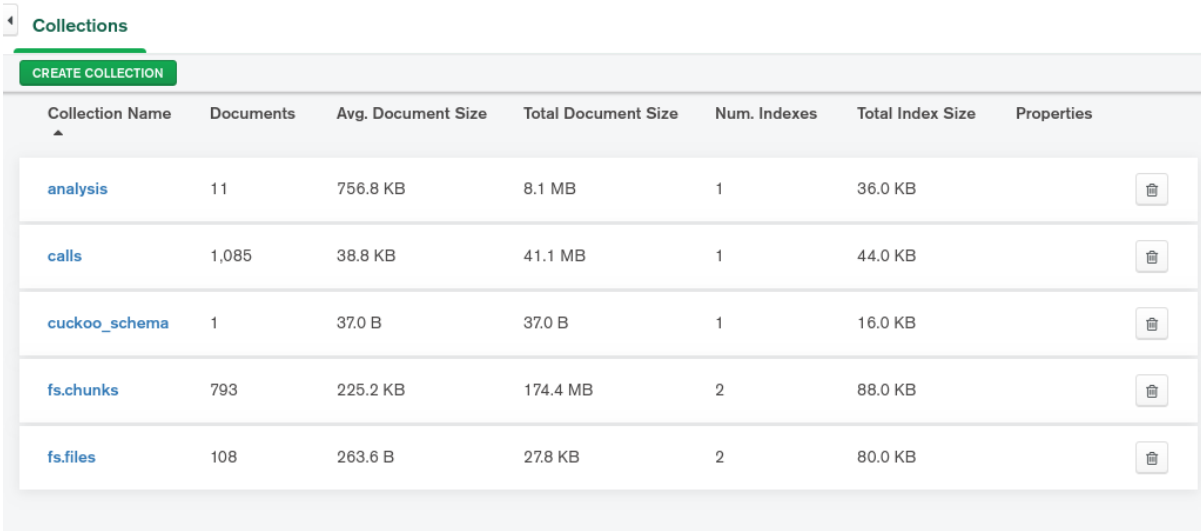


Figura 9 – Máquina virtual executando arquivo.

Os dados coletados pelo processo *cuckoo.exe* serão, por fim, salvos no MongoDB, dentro do *schema* Cuckoo, que contém as *collections* ilustradas na Figura 10.








Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
analysis	11	756.8 KB	8.1 MB	1	36.0 KB	
calls	1,085	38.8 KB	41.1 MB	1	44.0 KB	
cuckoo_schema	1	37.0 B	37.0 B	1	16.0 KB	
fs.chunks	793	225.2 KB	174.4 MB	2	88.0 KB	
fs.files	108	263.6 B	27.8 KB	2	80.0 KB	

Figura 10 – Collections do Schema Cuckoo.

4.2 Estrutura da Base de Dados

Em resumo, a coleção *analysis* é responsável por guardar os dados resultantes da fase de análise de maneira sintetizada. Porém também são criadas outras coleções, como *fs.chunks* que contém *chunks* da memória do sistema durante a execução do arquivo. Assim, caso seja necessário fazer um estudo mais aprofundado, esses dados podem ser consultados. Para a geração do conjunto de dados, foi considerada apenas a coleção de *analysis*, uma vez que um *chunk* inteiro da memória conteria muitas informações irrelevantes, enquanto que as *URLs* extraídas desse *chunk* (que estão contidas em *analysis*) podem ser de extrema importância. A estrutura dos dados contidos dentro da coleção pode ser vista nas Figuras 11 e 12.

```
_id: ObjectId("6125a3b10b66313dfe246874")  
> info: Object  
> procmemory: Array  
> target: Object  
> shots: Array  
> extracted: Array  
> signatures: Array  
> static: Object  
> dropped: Array  
> behavior: Object  
> debug: Object  
> metadata: Object  
> strings: Array  
> network: Object
```

Figura 11 – Estrutura dos dados pertencentes à *collection analysis*.

```

  info: Object
    added: 2021-08-24T22:55:32.703+00:00
    started: 2021-08-24T22:55:33.008+00:00
    duration: 142
    analysis_path: "/home/cuckoo/.cuckoo/storage/analyses/1"
    ended: 2021-08-24T22:57:55.643+00:00
    owner: null
    score: 4.2
    id: 1
    category: "file"
  git: Object
    head: "13cbe0d9e457be3673304533043e992ead1ea9b2"
    fetch_head: "13cbe0d9e457be3673304533043e992ead1ea9b2"
    monitor: "2deb9ccd75d5a7a3fe05b2625b03a8639d6ee36b"
    package: "exe"
    route: "none"
    custom: null
  machine: Object
    status: "stopped"
    name: "192.168.56.1011"
    label: "192.168.56.1011"
    manager: "VirtualBox"
    started_on: "2021-08-24 22:55:33"
    shutdown_on: "2021-08-24 22:57:55"
    platform: "windows"
    version: "2.0.7"
    options: "procmemdump=yes,route=none"

```

Figura 12 – Estrutura hierárquica dos dados do campo *info*, pertencente à *collection analysis*.

Como se pode ver, os dados da coleção estão salvos de maneira hierárquica, o que dificulta a transformação de *JSON* para *CSV*, e é exatamente aqui onde começam a surgir alguns problemas. Além da estrutura hierárquica dos dados, alguns campos são “infinitos”, como as *URLs* acessadas (Figura 13), já que diferentes arquivos podem acessar incontáveis *URLs* diferentes, assim dificultando ainda mais a representação desses dados no formato *CSV*— uma vez que geraria arquivos com milhares de colunas, o que não só prejudicaria o desempenho do algoritmo de aprendizado de máquina, como poderia causar também, o travamento da maioria dos programas que lessem o formato *CSV*. O mesmo problema ocorre com chamadas a *APIs*, *strings* contidas no programa, *DLLs* carregadas, assinaturas e etc. A solução encontrada para contornar tais problemas é detalhada na Seção 4.3.1.

```

  procmemory: Array
    0: Object
      regions: Array
      yara: Array
      num: 1
      file: "/home/cuckoo/.cuckoo/storage/analyses/1/memory/2288-1.dmp"
    urls: Array
      0: "http://www.microsoft.com/pki/certs/CSPCA.crt0"
      1: "http://support.microsoft.com/kb/2524860.KB2524860.ARP.DisplayNameDispl..."
      2: "http://microsoft.com0"
      3: "http://support.microsoft.com/kb/2524860KB2524860.ARP.NoModifyNoModifyK..."
      4: "http://support.microsoft.com/kb/2549743.KB2549743.ARP.DisplayNameHotfi..."
      5: "http://support.microsoft.com/kb/2544655KB2544655.ARP.NoModifyKB2544655..."
      6: "http://support.microsoft.com/kb/2565063KB2565063.ARP.NoModifyKB2565063..."
      7: "http://support.microsoft.com/kb/2549743KB2549743.ARP.NoModifyKB2549743..."
      8: "http://www.microsoft.com/pki/certs/tspca.crt0"
      9: "http://www.microsoft.com/MsiPatchSequenc"
      10: "http://support.microsoft.com/kb/2565063.KB2565063.ARP.DisplayNameHotfi..."
      11: "http://support.microsoft.comKB2524860.SE.ARPLinkARPLinkHKEY"
      12: "http://support.microsoft.com/kb/2544655.KB2544655.ARP.DisplayNameHotfi..."

```

Figura 13 – Exemplo de como ficam armazenadas as *URLs* acessadas por um arquivo.

4.3 Geração do conjunto de dados

Para que o usuário consiga gerar o conjunto de dados, adicionamos à barra de navegação o item “Dataset” (Figura 14), que, quando clicado, redirecionará o usuário para a página criada.

Como foi visto na seção referente ao Django, para que essa nova página possa ser “alcançável”, foi necessário adicionar seu caminho no arquivo `urls.py` (Figura 15), que é encarregado de direcionar as requisições à `view` correta. Posteriormente, foi adicionado um novo método na `view.py` adequada. Nesse caso, foi escolhida a `view` responsável pelos dados de análise (Figura 16) e, finalmente, a nova página foi adicionada à pasta `templates`, com o nome `dataset.html`, assim iniciando as etapas de desenvolvimento que serão descritas no decorrer dessa seção.

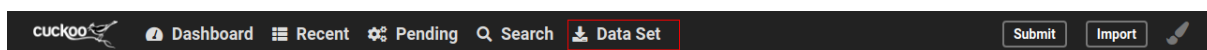


Figura 14 – Item adicionado a *Navbar* da plataforma *Cuckoo Web*.

```
urlpatterns = [
    url(r'^$', AnalysisRoutes.recent, name="analysis/recent"),
    url(r"^(?P<task_id>\d+)/$", AnalysisRoutes.redirect_default, name="analysis/redirect_default"),
    url(r"^(?P<task_id>\d+)/export/$", AnalysisRoutes.export, name="analysis/export"),
    url(r"^(?P<task_id>\d+)/reboot/$", SubmissionRoutes.reboot, name="analysis/reboot"),
    url(r"^(?P<task_id>\d+)/control/$", AnalysisControlRoutes.player, name="analysis/control/player"),
    url(r"^(?P<task_id>\d+)/control/screenshots/$", ControlApi.store_screenshots, name="analysis/control/screenshots"),
    url(r"^(?P<task_id>\d+)/control/tunnel/.*", ControlApi.tunnel, name="analysis/control/tunnel"),
    url(r"^(?P<task_id>\d+)/compare/$", AnalysisCompareRoutes.left, name="analysis/compare/left"),
    url(r"^(?P<task_id>\d+)/compare/(?P<compare_with_task_id>\d+)/$", AnalysisCompareRoutes.both, name="analysis/compare/both"),
    url(r"^(?P<task_id>\d+)/compare/(?P<compare_with_hash>.*)/$", AnalysisCompareRoutes.hash, name="analysis/compare/hash"),
    # TODO Get rid of this magic routing again as it's only complicating the URL routing.
    url(r"^(?P<task_id>\d+)/(?P<page>summary)$", AnalysisRoutes.detail, name="analysis"),
    url(r"^(?P<task_id>\d+)/(?P<page>\w+)/$", AnalysisRoutes.detail, name="analysis"),
    url(r"^(?P<task_id>\d+)/(?P<page>\w+)/$", AnalysisRoutes.detail, name="api"),
    url(r"^latest/$", views.latest_report),
    url(r"^remove/(?P<task_id>\d+)/$", views.remove),
    url(r"^chunk/(?P<task_id>\d+)/(?P<pid>\d+)/(?P<pagenum>\d+)/$", views.chunk),
    url(r"^filtered/(?P<task_id>\d+)/(?P<pid>\d+)/(?P<category>\w+)/$", views.filtered_chunk),
    url(r"^search/(?P<task_id>\d+)/$", views.search_behavior),
    url(r"^search/$", views.search),
    url(r"^pending/$", views.pending),
    url(r"^dataset/$", views.dataset),
]
```

Figura 15 – Linha de código incluída no arquivo `urls.py`.

```
@require_safe
def dataset(request):
    dataset_cols = "";

    return render_template(request, "analysis/dataset.html", **{
    })
```

Figura 16 – Trecho de código incluído no arquivo `analysis/views.py`.

4.3.1 Implementação Front-End

Para tratar os problemas com relação à estrutura dos dados do MongoDB citados anteriormente, os dados foram divididos em dois conjuntos, os dados unários, ou seja, os nós folhas da hierarquia, que possuem apenas um valor (*strings*, inteiros, reais), e os dados tipo lista, que não possuem valores padrões e nem um tamanho máximo definido, os quais são os casos de *URLs*, chamadas a *APIs*, etc.

No primeiro caso, para os dados unários, como visto na Figura 12, realizamos concatenações dos níveis de suas hierarquias, como por exemplo, o campo *name*, que é filho de *machine* e neto de *info*, foi transformado em uma coluna *CSV* de nome *info.machine.name*.

No *Front-end*, essa estrutura foi representada através da biblioteca *jsTree* (Figura 17), que é utilizada para criar estruturas de árvores, onde o usuário pode selecionar entre os campos que queira ou não, em seu conjunto de dados. Como esses tipos de dados estão sempre presentes nos *reports*, foi possível criar um solução geral para os mesmos.

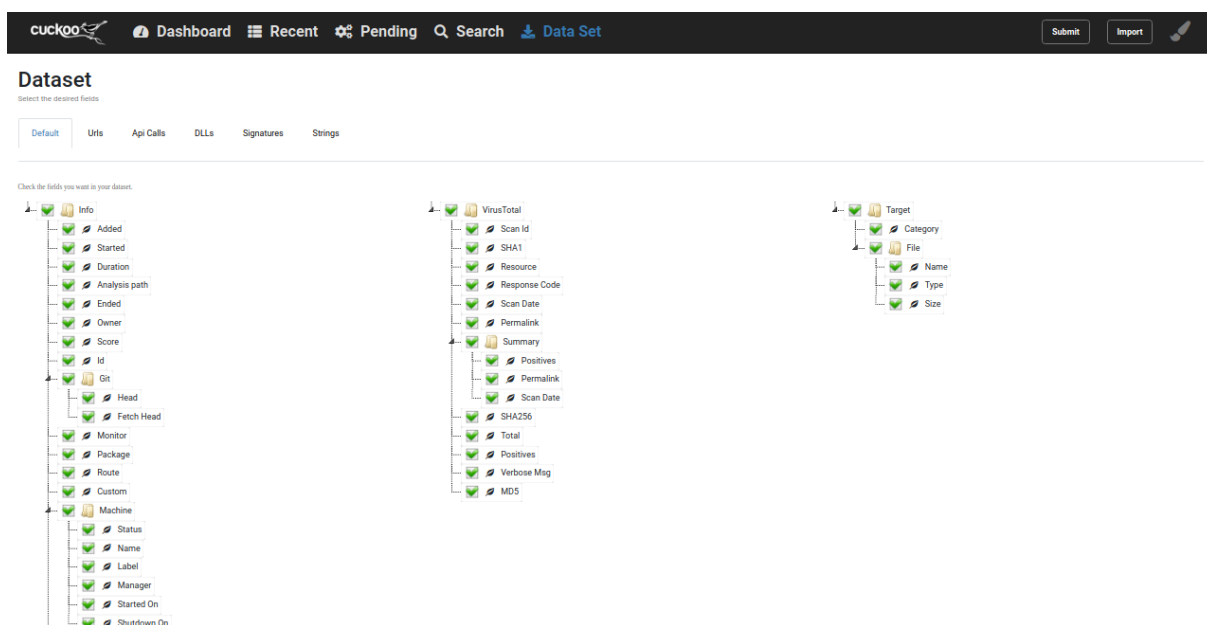


Figura 17 – Representação *Front-end* para campos unários.

Porém, para os dados do tipo lista, em que não há um limite de quantos campos existem, como no caso das *URLs*, foi necessário abrir mão de uma solução geral, para fornecer a maior flexibilidade possível, de modo que os pesquisadores informem as *URLs*, *APIs*, *DLLs*, assinaturas, *strings* que eles desejem incluir no conjunto de dados. Por exemplo, vamos supor que um pesquisador tenha interesse em incluir a *URL* <http://www.sitesuspeito.com> em seu *dataset*. Para que isso aconteça, basta que ele informe esse valor no campo correspondente a *URLs* no *Front-end*, e a adicione, assim a

plataforma irá gerar um conjunto de dados que conterà uma coluna com o nome deste site, e o valor de cada linha da tabela, nesta coluna, representará se tal arquivo acessou ou não tal URL.

No *Front-end*, a representação escolhida foi um *input* do tipo *text*, onde o usuário irá adicionar as *URLs*, *APIs* e etc (Figura 18 e 19). Para melhor diferenciação, esses dados foram divididos em abas, onde cada aba irá conter o *input* referente a ela.

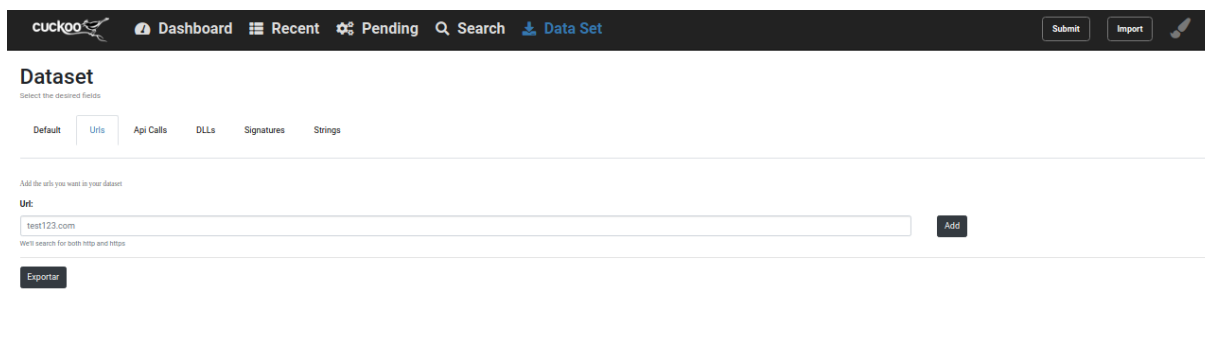


Figura 18 – Representação *Front-end* para campos do tipo lista, sem dados adicionados.

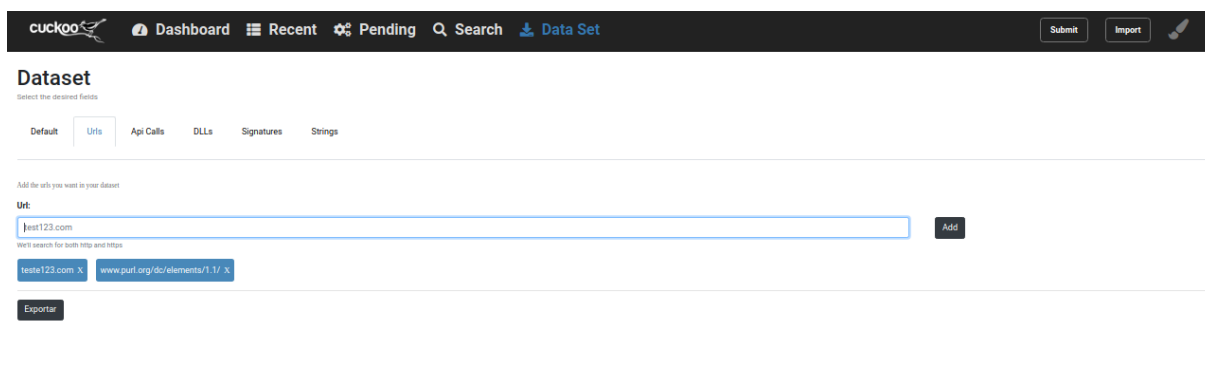


Figura 19 – Representação *Front-end* para campos do tipo lista, com dados adicionados.

Uma vez concluída a fase de seleção dos atributos, basta clicar no botão “exportar”, assim disparando um evento de clique (Figura 20), que basicamente irá coletar todos os dados que foram selecionados e adicionados pelo usuário, através das funções *get_tree_selected_values(tree_id)* (Figura 21), que percorre a estrutura de árvores, retornando o nome de cada campo que foi selecionado, como mostra a Figura(23), e da função *get_add_values(field_id)* (Figura 22), que irá coletar os nomes de *URLs*, *APIs* e *DLLs* adicionados pelo usuário, como mostra a Figura(24).

```
//Export
$("#export_dataset").on("click", (callback) =>{

    var info_nodes_checkedds = get_tree_selected_names("#jstree_info");
    var virus_total_nodes_checkedds = get_tree_selected_names("#jstree_virus_total");
    var target_nodes_checkedds = get_tree_selected_names("#jstree_target");

    var all_nodes_checkedds = info_nodes_checkedds.concat(virus_total_nodes_checkedds)
    .concat(target_nodes_checkedds);

    var params = {
        "task_id": task id,
        "selected_nodes": all_nodes_checkedds,
        "url_adds": get_adds("#url_adds"),
        "api_adds": get_adds("#api_adds"),
        "dlls_adds": get_adds("#dlls_adds"),
        "signature_adds": get_adds("#signature_adds"),
        "strings_adds": get_adds("#strings_adds")
    };

    CuckooWeb.api_post("/analysis/api/task/export_dataset/", params, function (data) {
        exportCSVFile(data.csv_string, "dataset");
    });
});
});
```

Figura 20 – Código do evento de clique do botão *Export*.

```
//FUNCTIONS
function get_tree_selected_names(tree_id){
    var selectedds = $(tree_id).jstree("get_selected", true);

    var parents;
    var parent_text;
    var final_text;
    var final_text_aux;

    var selectedds_names = [];
    for (var i = 0; i < selectedds.length; i++) {
        parents = $(selectedds[i].parents);

        final_text_aux = "";
        if(selectedds[i].children.length == 0) {
            for (var j = parents.length - 1; j >= 0; j--) {
                parent = parents[j];

                if (parent != "#") {
                    parent_text = $(tree_id).jstree(true).get_node(parent).text.trim().toLowerCase().replace(' ', '_');
                    final_text_aux += "." + parent_text;
                }
            }

            final_text = final_text_aux.substring(1, final_text_aux.length) + "." + selectedds[i].text.trim().toLowerCase();
            final_text = final_text.replace(' ', '_');

            selectedds_names.push(final_text);
        }
    }
    return selectedds_names
}
```

Figura 21 – Código da função `get_tree_selected_names()`.

```
function get_adds(add_id){
    itens_adds = $(add_id).children();

    itens_values = []
    for(var i = 0; i < itens_adds.length; i++){
        item_text = $(itens_adds[i]).text();

        itens_values.push(
            item_text.substring(0, item_text.length - 2)
        );
    }

    return itens_values;
}
```

Figura 22 – Código da função `get_adds()`.

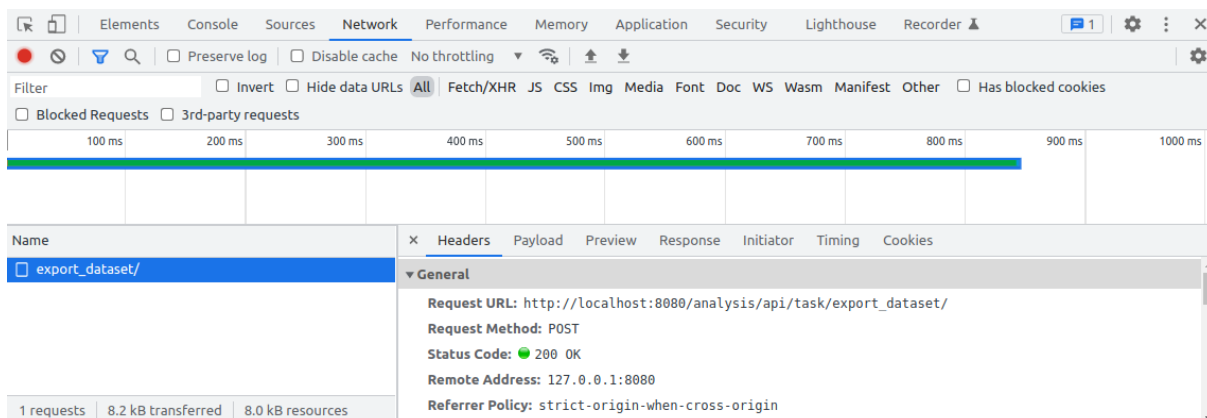
```
> get_tree_selected_names("#jstree_virus_total");
< (5) ['virustotal.summary.permalink', 'virustotal.summary.scan_date', 'virustotal.md5', 'virustotal.positives', 'virustotal.response_code']
  0: "virustotal.summary.permalink"
  1: "virustotal.summary.scan_date"
  2: "virustotal.md5"
  3: "virustotal.positives"
  4: "virustotal.response_code"
  length: 5
```

Figura 23 – Exemplo de retorno da função `get_tree_selected_names()`.

```
> get_adds("#url_adds");
< (2) ['teste123.com', 'www.purl.org/dc/elements/1.1/']
  0: "teste123.com"
  1: "www.purl.org/dc/elements/1.1/"
  length: 2
  ▶ [[Prototype]]: Array(0)
```

Figura 24 – Exemplo de retorno da função `get_adds()`.

Uma vez que os dados tenham sido coletados, uma requisição *ajax* do tipo *POST* é feita à *API* do *Cuckoo* (Figura 25), que irá processar os dados e montar o conjunto de dados como explicado na próxima subseção.

Figura 25 – Requisição *POST* feita à API do *Cuckoo Sandbox*.

4.3.2 Implementação Back-End

Uma vez que a requisição de exportação tenha sido recebida pelo servidor, inicia-se a etapa de processamento dos dados. Primeiramente, é realizada uma busca de todas as *rows* da *collection analysis* no MongoDB (Figura 26). Assim que a consulta a base de dados tenha retornado, inicia-se a montagem do arquivo *CSV*, percorrendo as *rows* e buscando os valores que o usuário selecionou na etapa anterior.

```
rows = mongo.db.analysis.find(
    {},
    ["info", "target", "behavior", "strings", "signatures", "virustotal", "procmemory"],
    sort=[("_id", pymongo.DESCENDING)]
)
```

Figura 26 – Código para buscar campos da *collection analysis*.

Para a forma mais simples de dados, os dados unários, o algoritmo (Figura 27) percorre a hierarquia do *JSON* até encontrar um nó folha. Caso esse nó folha possua o mesmo nome que algum dos nós que o usuário selecionou, o valor desse nó é adicionado em um dicionário (uma vez que cada *row* de um *dataframe* do *pandas* pode ser representada por uma estrutura de *chave-valor*, onde a chave corresponde ao nome da coluna e o valor representa seu valor naquela linha e naquela coluna). Fazendo esse mesmo processo para todos os campos, conseguimos então cobrir a geração dos dados unários.


```
# Populate Data Frame
data_frame = pd.DataFrame()
for row in rows:
    task = {}
    for selected_node in selected_nodes:
        hierarchy = selected_node.split('.')
        collection_name = hierarchy[0]
        collection = row.get(collection_name, {})

        hierarchy_iterator = collection
        for item in hierarchy[1:]:
            hierarchy_iterator = hierarchy_iterator.get(item, {})

        if hierarchy_iterator == {}:
            task[selected_node] = ''
        else:
            task[selected_node] = hierarchy_iterator
```

Figura 27 – Código para extração dos valores dos campos unários selecionados pelo usuário.

Já para os dados tipo lista, isso já foi um pouco mais complicado, pois alguns campos são salvos como listas de *strings*, outros como listas de objetos, assim necessitando de uma solução mais específica para cada caso. Mas a ideia geral para cada um deles se mantém a mesma, que é basicamente percorrer as listas onde esses dados estão salvos (Figura 13, por exemplo) e verificar a existência ou não das *URLs*, *APIs* e outros campos adicionados pelo usuário em tais listas. Caso a lista não contenha esse dado, significa que aquele arquivo não acessou tal campo. Portanto, o valor do mesmo no conjunto de dados será 0, caso contrário seu valor será 1. O algoritmo pode ser visto na Figura 28.

```

proc_memory_collection = row.get("procmemory", {})
for url_add in url_adds:
    url_exists = False
    for process in proc_memory_collection:
        for url in process.get("urls", ""):
            if (url_add.lower().strip() == url.lower().strip()) or ("http://" + url_add.lower().strip() == url.lower().strip()) or
               ("https://" + url_add.lower().strip() == url.lower().strip()) or ("www." + url_add.lower().strip() == url.lower().strip()) or
               ("http://www." + url_add.lower().strip() == url.lower().strip()) or ("https://www." + url_add.lower().strip() == url.lower().strip()):
                url_exists = True
                break
    task[url_add] = url_exists

behavior_collection = row.get("behavior", {})
for api_add in api_adds:
    api_exists = False
    processes_apis = behavior_collection.get("apistats", {})
    for process_name in processes_apis:
        process_apis_names = set(k.lower().strip() for k in processes_apis[process_name])
        if api_add.lower().strip() in process_apis_names:
            api_exists = True
            break
    task[api_add] = api_exists

for dll_add in dlls_adds:
    dll_exists = False
    dlls_loaded = behavior_collection.get("summary", {}).get("dll_loaded", {})
    for dll in dlls_loaded:
        dll_formatted = str(dll.split('\\')[-1]).lower().strip()
        dll_add_formatted = str(dll_add).lower().strip()
        if dll_add_formatted.endswith(".dll"):
            dll_add_formatted = dll_add_formatted[:-4]

        if (dll_add_formatted == dll_formatted) or (dll_add_formatted + ".dll" == dll_formatted):
            dll_exists = True
            break
    task[dll_add] = dll_exists

signatures_collection = row.get("signatures", {})
for signature_add in signature_adds:
    signature_exists = False
    for signature in signatures_collection:
        if signature_add.lower().strip() == signature.get("name", '').lower().strip():
            signature_exists = True
            break
    task[signature_add] = signature_exists

strings_collection = row.get("strings", {})
for string_add in strings_adds:
    string_exists = False
    for string in strings_collection:
        if string_add.lower().strip() == string.lower().strip():
            string_exists = True
            break
    task[string_add] = string_exists

data_frame = data_frame.append(task, ignore_index=True)
return JsonResponse({"csv_string": data_frame.to_csv()}, safe=False)

```

Figura 28 – Código para extração dos valores dos campos do tipo lista adicionados pelo usuário.

Ao final de todas as iterações, teremos um *dataframe* que é o conjunto de todos os dicionários gerados, ou seja, de todas os arquivos no MongoDB. Como a *API* retorna apenas dados no formato *JSON* para o *Front-end*, o *dataframe* é retornado como um campo do *JSON* que será retornado, de nome *csv_string*, que conterà o arquivo *CSV*, em forma de *string* (Figura 29).

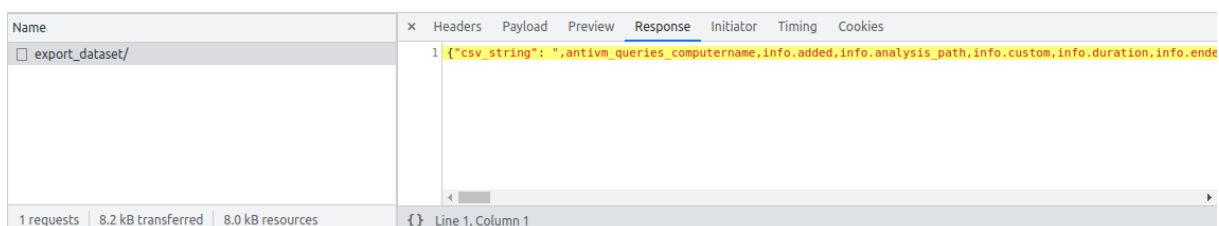


Figura 29 – Resposta a requisição feita a *API* do *Cuckoo Web*.

4.3.3 Download do conjunto de dados

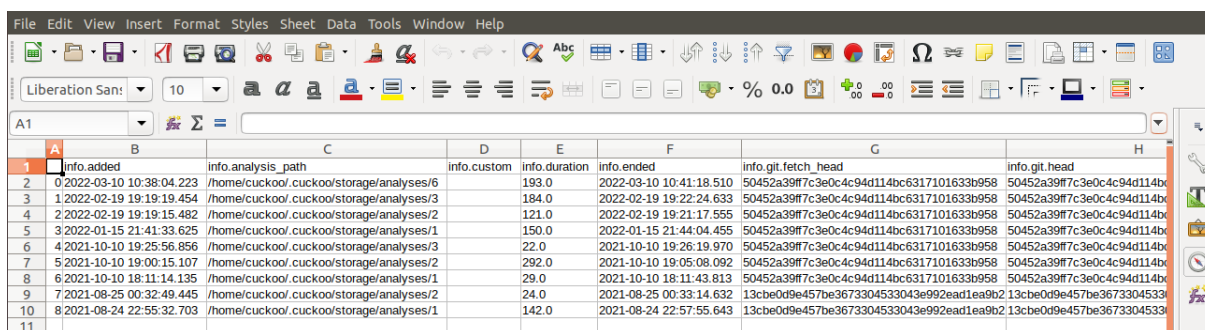
No *Front-end* por sua vez, caso o processamento tenha sido realizado com sucesso, utilizando-se da response gerada pela *API* (Figura 29), é criado um *Blob* (Binary Large Object), que nada mais é que um objeto Javascript que armazena uma sequência de bytes, contendo os dados do arquivo *CSV* gerado (Figura 30). Por fim é criada uma *URL* que referencia tal objeto e, através dela, é realizado o download do arquivo final (Figura 31)

```
function exportCSVFile(csv, fileTitle) {
    var exportedFilename = fileTitle + '.csv' || 'export.csv';
    var blob = new Blob([csv], { type: 'text/csv;charset=utf-8;' });

    if (navigator.msSaveBlob) { // IE 10+
        navigator.msSaveBlob(blob, exportedFilename);
    }
    else {
        var link = document.createElement("a");

        if (link.download !== undefined) {
            var url = URL.createObjectURL(blob);
            link.setAttribute("href", url);
            link.setAttribute("download", exportedFilename);
            link.style.visibility = 'hidden';
            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);
        }
    }
}
```

Figura 30 – Código para download do arquivo *CSV* gerado.



	A	B	C	D	E	F	G	H
1		info.added	info.analysis_path	info.custom	info.duration	info.ended	info.git_fetch_head	info.git.head
2	0	2022-03-10 10:38:04.223	/home/cuckoo/cuckoo/storage/analyses/6		193.0	2022-03-10 10:41:18.510	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
3	1	2022-02-19 19:19:19.454	/home/cuckoo/cuckoo/storage/analyses/3		184.0	2022-02-19 19:22:24.633	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
4	2	2022-02-19 19:19:15.482	/home/cuckoo/cuckoo/storage/analyses/2		121.0	2022-02-19 19:21:17.555	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
5	3	2022-01-15 21:41:33.625	/home/cuckoo/cuckoo/storage/analyses/1		150.0	2022-01-15 21:44:04.455	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
6	4	2021-10-10 19:25:56.856	/home/cuckoo/cuckoo/storage/analyses/3		22.0	2021-10-10 19:26:19.970	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
7	5	2021-10-10 19:00:15.107	/home/cuckoo/cuckoo/storage/analyses/2		292.0	2021-10-10 19:05:08.092	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
8	6	2021-10-10 18:11:14.135	/home/cuckoo/cuckoo/storage/analyses/1		29.0	2021-10-10 18:11:43.813	50452a39ff7c3e0c4c94d114bc6317101633b958	50452a39ff7c3e0c4c94d114bc6317101633b958
9	7	2021-08-25 00:32:49.445	/home/cuckoo/cuckoo/storage/analyses/2		24.0	2021-08-25 00:33:14.632	13cbe0d9e457be3673304533043e992ead1ea9b2	13cbe0d9e457be3673304533043e992ead1ea9b2
10	8	2021-08-24 22:55:32.703	/home/cuckoo/cuckoo/storage/analyses/1		142.0	2021-08-24 22:57:55.643	13cbe0d9e457be3673304533043e992ead1ea9b2	13cbe0d9e457be3673304533043e992ead1ea9b2
11								

Figura 31 – Exemplo de conjunto de dados, gerado no formato *CSV*.

5 Conclusões

Como se pôde ver, são muitos os desafios enfrentados por pesquisadores da área de segurança da informação, que é uma área em constante evolução e que está se desenvolvendo muito rapidamente. Os códigos maliciosos estão cada vez mais sofisticados, e *malwares* metamórficos são cada vez mais comuns, o que gera uma pressão enorme nos profissionais dessa área, que vivem nessa constante corrida de gato e rato contra os criminosos.

Uma possível vantagem que esses profissionais podem ter sobre os criminosos é a utilização da inteligência artificial como sua aliada, o que por si só já é um grande desafio, uma vez que para isso é necessário mais que apenas o estudo de métodos de aprendizado de máquina. Aqui se torna também necessária a criação de bons conjuntos de dados (algo que é extremamente complicado, uma vez que *malwares* metamórficos são extremamente difíceis de serem analisados), estudos específicos do domínio de cada tipo de *malware*, assim como constante atualização de conhecimento, uma vez que novos ataques surgem com frequência.

Este trabalho consegue aliviar um pouco as dores vividas por pesquisadores da área, tornando possível a criação de conjuntos de dados personalizados para cada domínio de pesquisa, o que facilitará futuros estudos, já que será possível testar modelos de aprendizado de máquina, utilizando diferentes conjuntos de dados.

A ferramenta proposta fornece também uma plataforma unificada que conterá diversos códigos maliciosos já analisados, fazendo com que não seja necessário o retrabalho de busca de tais arquivos, além de servir como um arcabouço para futuros projetos, podendo ser adicionados métodos para evitar *sandbox evasion*, melhorando assim a qualidade do conjuntos de dados, criação de *Web crawlers*, para continuamente povoar a base de dados, incluir novos dados de *report* a base, entre diversas outras possibilidades.

Dessa forma, podemos ver a criação de tal plataforma como um “pequeno grande passo”, para a criação de uma estrutura, onde diversos pesquisadores possam se beneficiar e avançar seus estudos, colaborando para que um dia a inteligência artificial mitigue de forma significativa os estragos causados por criminosos cibernéticos.

Referências

- CATAK, F. et al. Deep learning based sequential model for malware analysis using windows exe api calls. p. 23, 07 2020. DOI: <<https://doi.org/10.7717/peerj-cs.285>>. Citado 2 vezes nas páginas 12 e 14.
- CERT.BR. *Cartilha de segurança para Internet*. 2017. <<https://cartilha.cert.br/malware/>>. [Online; accessed 03-May-2021]. Citado na página 9.
- CHAILYTKO, A.; SKURATOVICH, S. Defeating sandbox evasion: How to increase the successful emulation rate in your virtual environment. In: . Denver, CO: Virus Bulletin, 2016. <<https://www.virusbulletin.com/uploads/pdf/magazine/2016/VB2016-Chailtyko-Skuratovich.pdf>>. Citado na página 13.
- DOUZI, S. et al. Hybrid email spam detection model using artificial intelligence. *International Journal of Machine Learning and Computing*, v. 10, p. 316–322, 02 2020. DOI: <<https://doi.org/10.18178/ijmlc.2020.10.2.937>>. Citado na página 10.
- EHTESHAMIFAR, S. et al. Easy to fool? testing the anti-evasion capabilities of pdf malware scanners. p. 14, 01 2019. DOI: <<https://doi.org/10.48550/arXiv.1901.05674>>. Citado 3 vezes nas páginas 12, 13 e 14.
- FERRAND, O. How to detect the cuckoo sandbox and to strengthen it? *Journal of Computer Virology and Hacking Techniques*, p. 08, 02 2015. DOI: <<https://doi.org/10.1007/s11416-014-0224-9>>. Citado 2 vezes nas páginas 13 e 14.
- GUIBERNAU, F. Catch me if you can!—detecting sandbox evasion techniques. In: . San Francisco, CA: USENIX Association, 2020. <<https://www.usenix.org/conference/enigma2020/presentation/guibernau>>. Citado na página 14.
- MILLER, C. et al. Insights gained from constructing a large scale dynamic analysis platform. p. 09, 09 2017. DOI: <<https://doi.org/10.1016/j.diin.2017.06.007>>. Citado na página 14.
- RAD, B.; MASROM, M.; IBRAHIM, S. Camouflage in malware: from encryption to metamorphism. p. 11, 08 2012. <https://www.researchgate.net/publication/235641122_Camouflage_In_Malware_From_Encryption_To_Metamorphism>. Citado 2 vezes nas páginas 12 e 14.
- SHAKYA, S. Analysis of artificial intelligence based image classification techniques. *Journal of Innovative Image Processing*, p. 11, 03 2020. DOI: <<https://doi.org/10.36548/jiip.2020.1.005>>. Citado na página 10.
- YUSIRWAN, S.; PRAYUDI, Y.; RIADI, I. Implementation of malware analysis using static and dynamic analysis method. *International Journal of Computer Applications*, v. 117, p. 975–8887, 04 2015. DOI: <<http://dx.doi.org/10.5120/20557-2943>>. Citado 3 vezes nas páginas 9, 11 e 13.