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

Guilherme V. Figueiredo[1], **Renan G. Cattelan**[1], **Rodrigo S. Miani**[1]

[1]Faculdade de Computação – Universidade Federal de Uberlândia (UFU)
Caixa Postal 38408-100 – Uberlândia – MG – Brazil

{guilhermefigueiredo, renan, miani}@ufu.br

Abstract. *This paper seeks to create a unified platform for dynamic and static* malware *analysis, in which several researchers can generate datasets for use in their research. It is proposed that the generation of such datasets be flexible enough to cover the different areas of* malware *research, allowing choosing among the fields obtained from the analysis phase, which may or may not be included in the resulting dataset. Through extensions to the functionalities of the* Cuckoo Sandbox *platform, new functionalities were implemented, allowing the batch download of reports generated in* CSV *format, more suitable for machine learning algorithms.*

Resumo. *Este trabalho 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* Cuckoo Sandbox*, foram implementadas novas funcionalidades, permitindo o download em lote dos relatórios gerados em formato* CSV*, mais adequado para algoritmos de aprendizado de máquina.*

1. Introdução

Malwares, ou códigos maliciosos, são softwares projetados intencionalmente para causar danos aos sistemas de computador e comprometer a segurança dos usuários [Catak et al. 2020]. Assim, para melhor compreender e permitir combater esse tipo de ameaça, surge a área de análise de *malware*, que estuda como códigos maliciosos funcionam, aplicando diversas técnicas para obter tal resultado. As duas principais técnicas utilizadas para realização de tais análises, são [Yusirwan et al. 2015]:

- Análise estática: que é um método de análise realizada sem a execução do *malware*, onde ocorre a verificação por antivírus, *hash* ou outros atributos do arquivo.
- Análise dinâmica: onde a análise é feita executando-se o *malware* em uma máquina virtual, para tornar a análise mais segura, onde ocorre então o monitoramento dos pacotes enviados, *system calls*, registros modificados, chamadas a *APIs*, entre outros atributos de execução.

Nesse contexto, têm-se também os antivírus, que são *softwares* criados com o intuito de detectar *malwares* no momento em que eles tentam infectar um sistema. Basicamente, eles utilizam um mecanismo de assinaturas, pelo qual conseguem verificar, em uma base 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 e que estejam presentes em suas bases de dados. Então, mesmo que o antivírus esteja instalado, é possível que o sistema seja infectado por um vírus recente, razão pela qual as bases de dados sofrem constantes atualizações, com a adição de novas ameaças conforme são descobertas.

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, torna-se de extremo interesse obter uma ferramenta que permita detectar *malwares* antes dos mesmos serem conhecidos. Assim, utilizando-se de técnicas de aprendizado de máquina, seria possível atingir tal objetivo [Gibert et al. 2020].

Porém, para que seja possível a utilização de tais técnicas, se faz necessário o uso de conjuntos de dados (*datasets*) criados a partir dos dados obtidos via análises 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 ou na análise, propriamente dita, dos seus atributos principais.

O objetivo deste trabalho foi desenvolver uma plataforma que padronizasse o ambiente de análise e o carregamento dos conjuntos de dados como um serviço computacional (*Sandbox as a Service*) capaz de 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 tal, foi feito uso da plataforma *Cuckoo Sandbox* [Cuckoo Foundation 2010], sobre a qual foram realizadas diversas configurações, bem como extensões em suas funcionalidades, de modo que esta passasse a disponibilizar uma sessão onde fosse possível gerar conjuntos de dados personalizados contendo dados obtidos a partir de análise estática e dinâmica de todos os arquivos que já foram analisados por ela, abrindo assim uma gama de novas possibilidades para o grupo de pesquisa e permitindo-lhe focar na análise dos dados em si, sendo os detalhes técnicos simplificados.

O restante do texto está organizado da seguinte forma: a Seção 2 discute trabalhos relacionados; a Seção 3 descreve as principais ferramentas que serviram de base para fundamentação da proposta; a Seção 4 apresenta a proposta em si; e, por fim, a Seção 5 traz as conclusões do trabalho.

2. Trabalhos Relacionados

[Miller et al. 2017] discorrem sobre os conhecimentos obtidos ao se 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 decisões de projeto e escolhas de implementação.

[Borges et al. 2021] descrevem o processo de construção de um conjunto de dados para análise estática de *ransomwares*, evidenciando a necessidade de dados para melhor compreender e permitir combater esse tipo de ameaça.

[Yusirwan et al. 2015] utilizaram 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.

[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 oito tipos diferentes de *malware* (*Adwares*, *Trojans*, *Spywares*, etc.) e obtém uma boa acurácia de classificação em todos.

[Ehteshamifar et al. 2019] testam diversos analisadores de *malware*, para observar quais têm as melhores defesas contra técnicas de evasão. O 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 de forma separada 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. Da mesma forma, [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 contramedidas para tais técnicas.

[Sethi et al. 2018] propõem um *framework* para detectar e classificar diferentes arquivos como benignos ou maliciosos usando um classificador de dois níveis. Sua solução usa Cuckoo Sandbox para gerar um relatório de análise estática e dinâmica, executando os arquivos de amostra no ambiente virtual, bem como um módulo de extração de atributos que usa os relatórios gerados pelo Cuckoo Sandbox e emprega Weka para desenvolver modelos de aprendizado de máquina.

A proposta ora apresentada automatiza estratégias como as citadas, facilitando a criação de conjuntos de dados personalizados, onde é possível selecionar os atributos de interesse dentre todos os disponíveis nos relatórios gerados, inclusive em lote, e seu posterior emprego em formato padronizado e adequado aos *pipelines* de aprendizado de máquina comumente disponíveis.

3. Fundamentação Técnica

Para a realização das análises propostas, são necessárias diversas ferramentas, sendo as principais delas detalhadas ao longo desta seção.

3.1. Cuckoo Sandbox

O *Cuckoo Sandbox* [Cuckoo Foundation 2010] 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*, que contém o "núcleo" do *Cuckoo*. É ele que é responsável por receber as requisições, gerenciar o processamento de análises e dos diversos *guests*, que são ambientes isolados onde os arquivos enviados pelos usuários serão executados e analisados (Figura 1). Os usuários são capazes de analisar seus arquivos simultaneamente, dependendo da capacidade de processamento do servidor utilizado.

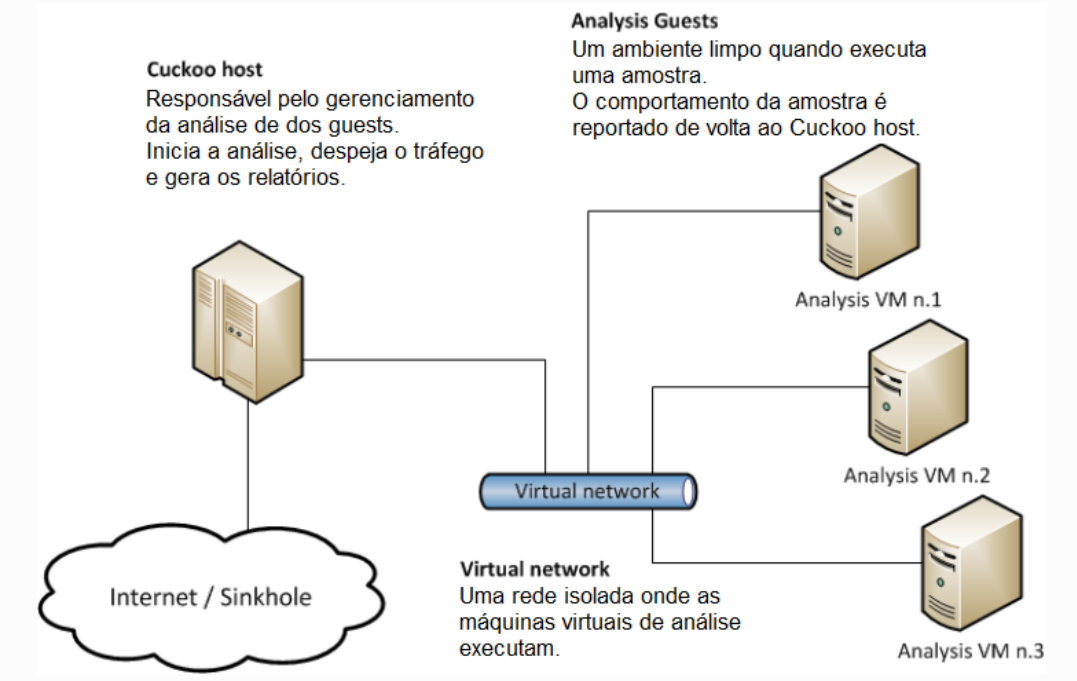


Figura 1. Arquitetura do *Cuckoo Sandbox* (adaptada de [Cuckoo Foundation 2010]).

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. Este 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 e Javascript. Já os dados da análise são salvos como documentos no MongoDB. Detalhes dessas etapas são apresentados na Seção 4.

Além disso, durante a fase de análise, é possível utilizar diversas ferramentas externas, conforme o que se deseja investigar. Uma das ferramentas utilizadas neste trabalho foi a *API* do *VirusTotal*, para que, além dos dados de análise dinâmica gerados pelo *Cuckoo Sandbox*, fosse possível incluir no *dataset* final também dados de análise estática.

3.2. VirusTotal

O *VirusTotal*[1] é uma base de dados, voltada para pesquisa, que contém diversas amostras de *malware* disponíveis para download, além da implementação de uma interface para análise estática. Mais especificamente, trata-se de uma *API REST* contendo diversos *endpoints* que tornam possível a realização de análises estáticas de arquivos através de

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

chamadas à *API*. Basicamente, os *endpoints* utilizados pelo *Cuckoo Sandbox* no contexto deste trabalho são:

- **api/v3/files:** *endpoint* por onde são enviados os binários de um arquivo para o servidor do *VirusTotal* e cuja resposta dessa requisição vem na forma de um arquivo *JSON* contendo principalmente o id, através do qual é possível resgatar a análise realizada sobre o arquivo enviado.
- **api/v3/analyses/id:** *endpoint* através de cujo id é possível recuperar os dados da análise estática representados por ele.

4. Proposta

Atualmente, a plataforma Web do *Cuckoo Sandbox* não disponibiliza nenhuma maneira para realizar download em lote dos dados[2] 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 – algo que, além de não ser trivial, 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*[3] e das tecnologias anteriormente citadas, foram implementadas novas funcionalidades na plataforma *Cuckoo Web*, de modo que se tornasse possível gerar os conjuntos de dados desejados já no formato *CSV*, apropriado para manipulação por ferramentas de aprendizado de máquina.

Como os pesquisadores, dependendo de sua linha de pesquisa, possuem diferentes requisitos e necessidades[4] para seu conjunto de dados, 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 dos *datasets* resultantes serão descritos nesta seção, assim como o que foi necessário para a implementação dessas novas funcionalidades. Todo o código-fonte modificado está disponível em `https://github.com/GuilhermeVF/Cuckoo_Modified_DatasetGenerator`.

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

[2]É possível fazer downloads de relatórios (*reports*) de arquivos específicos, no formato *JSON*, o que não é ideal para modelos de aprendizado de máquina.

[3]https://github.com/cuckoosandbox/cuckoo

[4]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.

à página inicial (Figura 2), que contém diversos dados, como análises recentes, poder de processamento disponível, entre outras informações. Clicando em "SUBMIT A FILE FOR ANALYSIS", destacado em vermelho na figura, é possível selecionar os arquivos que serão analisados.

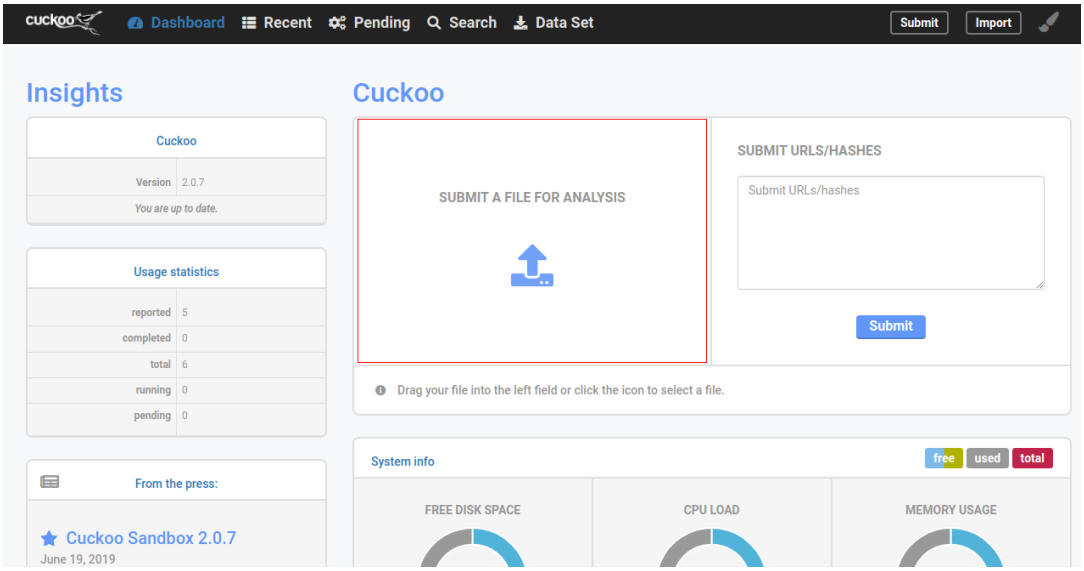


Figura 2. Página inicial da plataforma Cuckoo Web.

Uma vez selecionados, os arquivos serão mostrados na tela (Figura 3), assim como algumas opções da fase de análise, como *timeout*, tipo de roteamento de rede, etc.

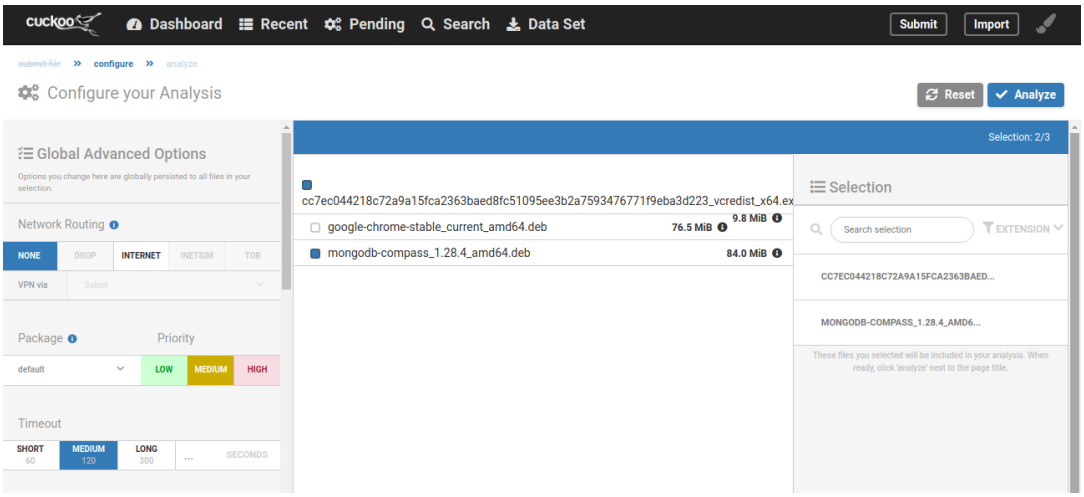


Figura 3. Página de configuração de opções de análise.

Terminada 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 4) até que a mesma seja concluída.

Uma vez que a requisição chegue ao servidor, são iniciadas máquinas virtuais (Figura 5), que serão responsáveis pela execução dos arquivos enviados. Essas máquinas

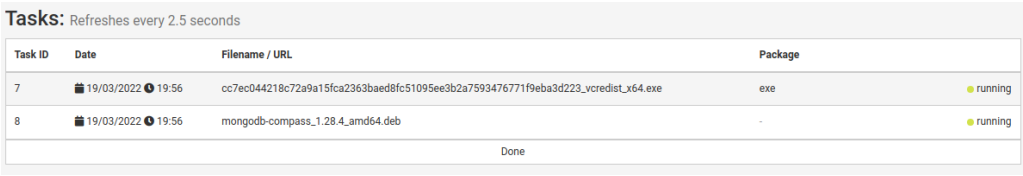


Figura 4. Página de exibição do status de processamento dos arquivos enviados.

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*. 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.

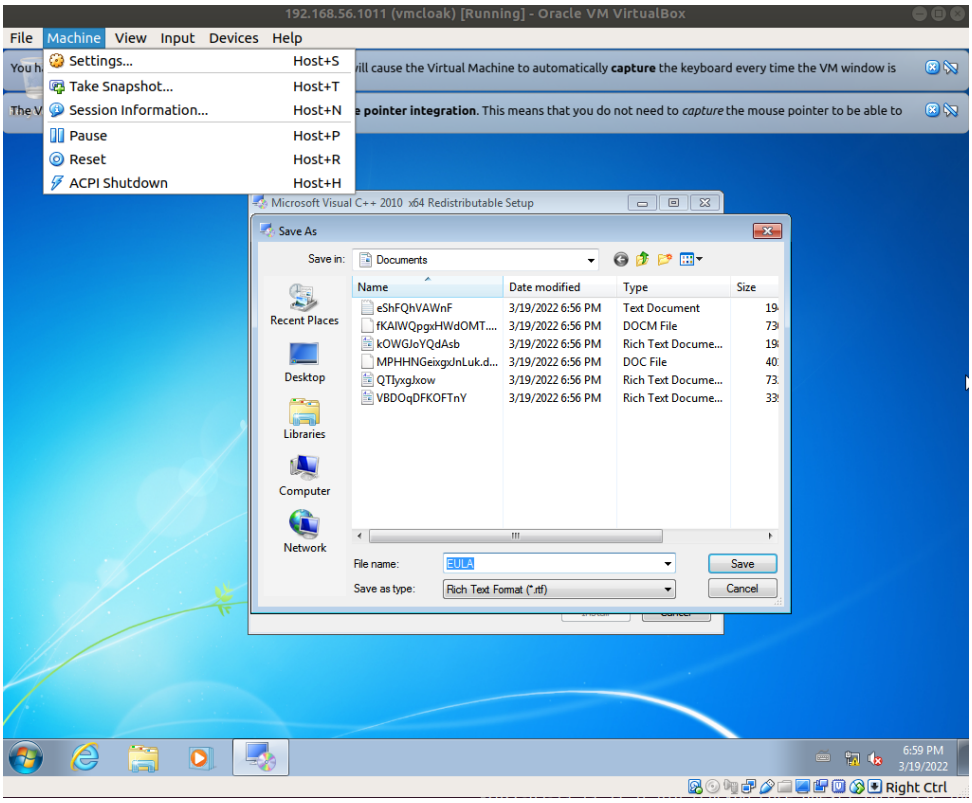


Figura 5. Máquina virtual executando arquivo.

Os dados coletados pelo processo *cuckoo.exe* serão, por fim, salvos no MongoDB, dentro do *schema* Cuckoo, que contém as *collections* ilustradas na Figura 6.

4.2. Estrutura da Base de Dados

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

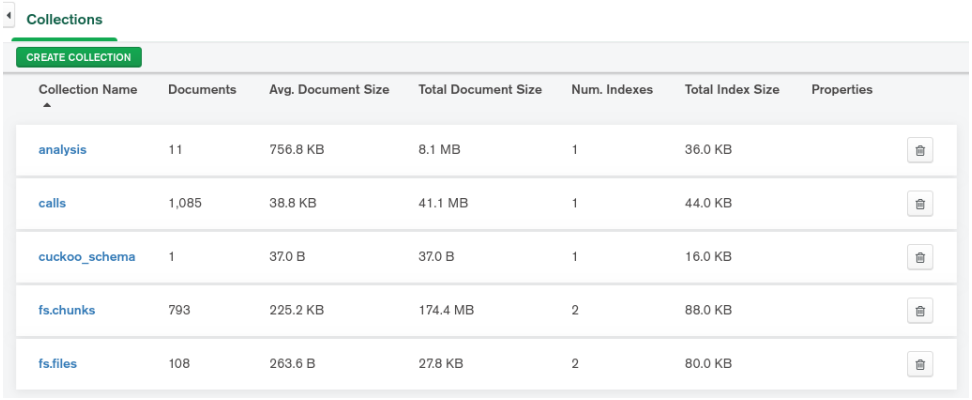


Figura 6. *Collections* do *Schema* Cuckoo.

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 na Figura 7.

Figura 7. Estrutura dos dados pertencentes à *collection analysis* (a), com detalhe para a estrutura hierárquica dos dados do campo *info* (b).

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 sãode tamanho indeterminado , como as *URLs* acessadas , já que diferentes arquivos podem acessar incontáveis *URLs* diferentes, dificultando assim ainda mais a representação desses dados no formato *CSV* – uma vez que isso 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, etc. A solução encontrada para contornar tais problemas é detalhada na Seção 4.3.1.

4.3. Geração do conjunto de dados

Para que o usuário consiga gerar o conjunto de dados, foi adicionado à barra de navegação o item "Dataset" (Figura 8), que, quando clicado, redireciona o usuário para a página criada. Para que essa nova página pudesse ser "alcançavel", foi necessário adicionar seu caminho no arquivo *urls.py*, 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 e, finalmente, a nova página foi adicionada à pasta *templates*, com o nome *dataset.html*, assim iniciando as etapas de desenvolvimento que serão descritas a seguir.

Figura 8. Item adicionado à *Navbar* da plataforma *Cuckoo Web*.

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 7(b) , realizamos concatenações dos níveis de suas hierarquias. 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 9), que é utilizada para criar estruturas de árvores, onde o usuário pode selecionar os campos desejados em seu conjunto de dados. Assim, como esses tipos de dados estão sempre presentes nos relatórios, foi possível criar uma solução geral.

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 maior flexibilidade, de modo que os pesquisadores pudessem informar as *URLs*, *APIs*, *DLLs*, assinaturas e *strings* desejadas. Por exemplo, vamos supor que um pesquisador tenha interesse em incluir a *URL* `http://www.teste123.com` em seu *dataset*. Para que isso aconteça, basta que ele informe esse valor no campo correspondente a *URLs* no *Front-end*. Assim, a plataforma irá gerar um conjunto de dados que conterá uma coluna com o nome do site desejado e o valor de cada linha da tabela, nessa coluna, representará se tal arquivo acessou ou não a URL dada.

No *Front-end*, a representação escolhida foi um *input* do tipo *text*, onde o usuário pode adicionar as informações desejadas (*URLs*, *APIs*, etc.), como ilustrado na Figura 10. Para melhor diferenciação, esses dados foram divididos em abas, onde cada aba contém o *input* referente a ela.

Uma vez concluída a fase de seleção dos atributos, basta clicar no botão "exportar", disparando assim um evento de clique, que basicamente irá coletar todos os dados que foram selecionados e adicionados pelo usuário, através das funções

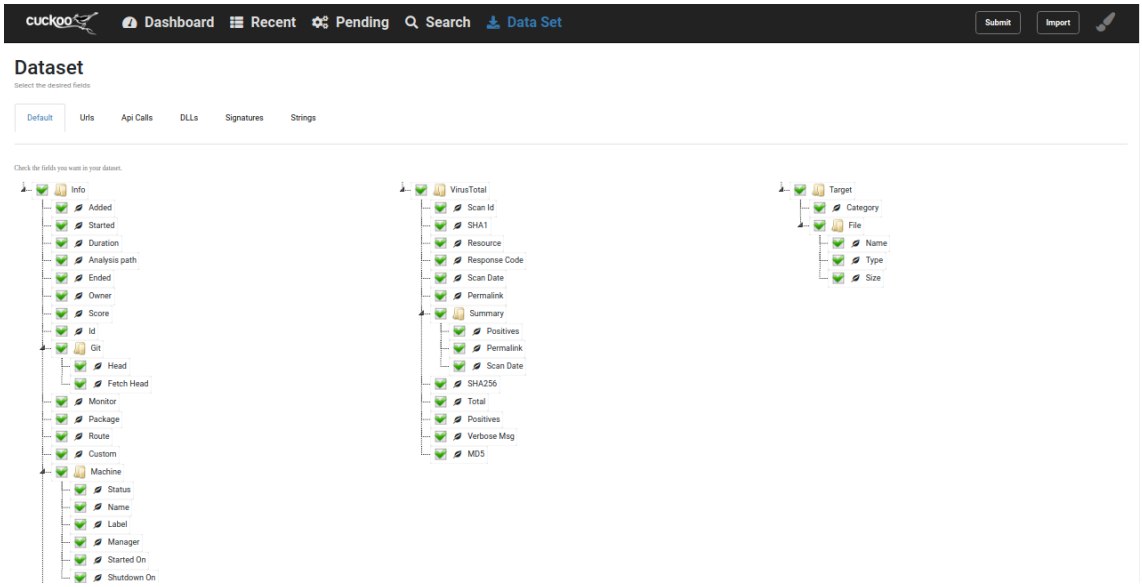


Figura 9. Representação *Front-end* para campos unários.

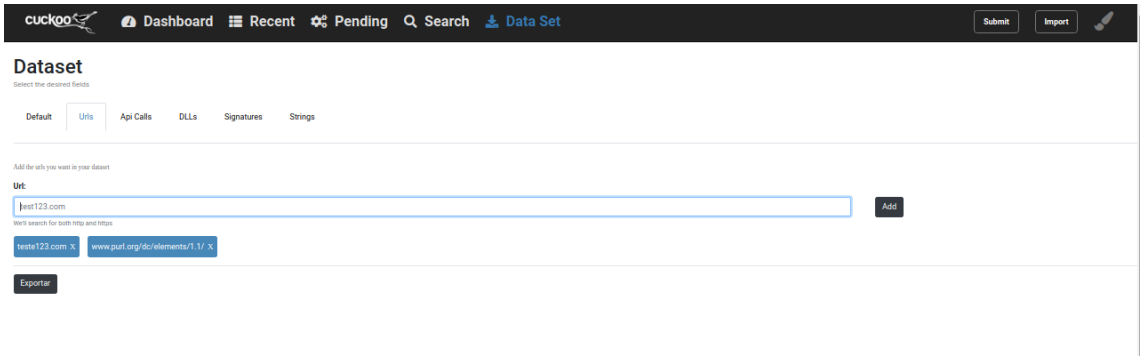


Figura 10. Representação *Front-end* para campos do tipo lista, com dados adicionados.

get_tree_selected_values(tree_id) (Figura 11), que percorre a estrutura de árvores, retornando o nome de cada campo que foi selecionado (Figura 12), e da função *get_add_values(field_id)* (Figura 13), que irá coletar e retornar os nomes de *URLs*, *APIs* e *DLLs* adicionados pelo usuário (Figura 14). Após a coleta dos dados, uma requisição *ajax* do tipo *POST* é feita à *API* do *Cuckoo*, que irá processar os dados e montar o conjunto de dados como explicado na próxima subseção.

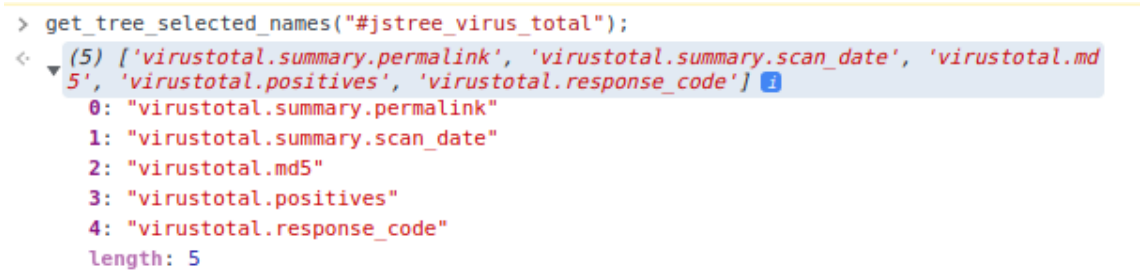
4.3.2. Implementação Back-End

Com o recebimento da requisição de exportação pelo servidor, inicia-se a etapa de processamento dos dados. Primeiramente, é realizada uma busca de todas as *rows* da *collection analysis* no MongoDB. 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.

Para os dados unários, o algoritmo apresentado na Figura 15 percorre a hierarquia

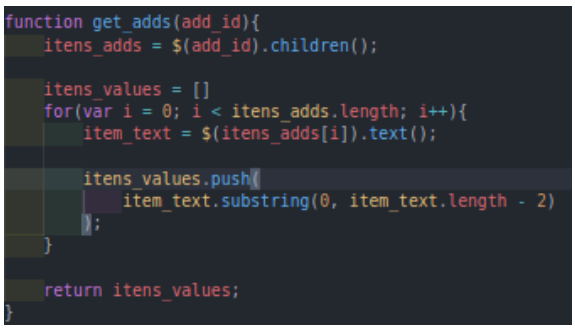
```
//FUNCTIONS
function get_tree_selected_names(tree_id){
    var selecteds = $(tree_id).jstree("get_selected", true);

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

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

        final_text_aux = "";
        if(selecteds[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) + "." + selecteds[i].text.trim().toLowerCase();
            final_text = final_text.replace(' ', '_')

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

Figura 11. Código da função get_tree_selected_names().



```
> get_tree_selected_names("#jstree_virus_total");
  (5) ['virustotal.summary.permalink', 'virustotal.summary.scan_date', 'virustotal.md
  5', '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 12. Exemplo de retorno 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 13. Código da função get_adds().

```
> 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 14. Exemplo de retorno da função get_adds().

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 *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, foi possível então cobrir a geração dos dados unários.

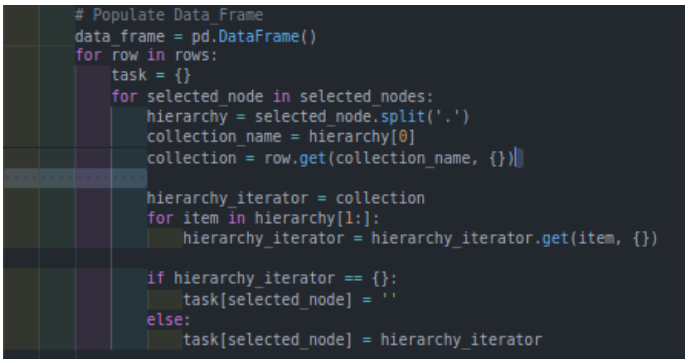


Figura 15. Código para extração dos valores dos campos unários selecionados pelo usuário.

Já para os dados do tipo lista, o processo é um pouco mais complicado, pois alguns campos são salvos como listas de *strings*, outros como listas de objetos, necessitando assim 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 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 16.

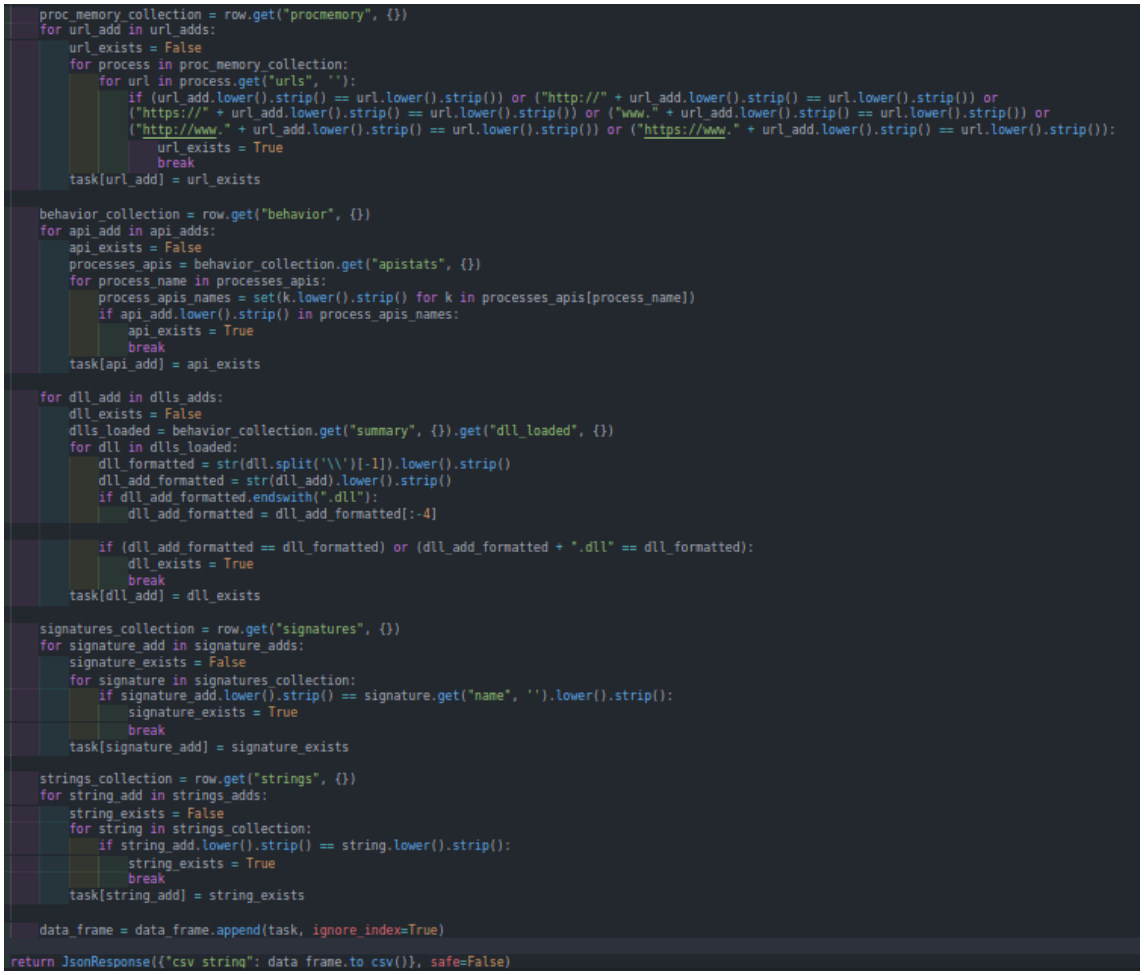
Ao final de todas as iterações, teremos um *dataframe* que é o conjunto de todos os dicionários gerados, ou seja, de todas os arquivos no MongoDB. Como a *API* retorna apenas dados no formato *JSON* para o *Front-end*, o *dataframe* é retornado como um campo do *JSON* que será retornado, de nome *csv_string*, que conterá o arquivo *CSV*, em forma de *string* (Figura 17).

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 resposta gerada pela *API* (Figura 17), 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. Por fim, é criada uma *URL* que referencia tal objeto e, através dela, é realizado o download do arquivo final (Figura 18).

5. Conclusão

Como se pôde ver, são muitos os desafios enfrentados por pesquisadores da área de segurança da informação. Os códigos maliciosos estão cada vez mais sofisticados e *malwares* metamórficos têm se tornado comuns, o que gera pressão para o uso de técnicas



```
    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 16. Código para extração dos valores dos campos do tipo lista adicionados pelo usuário.

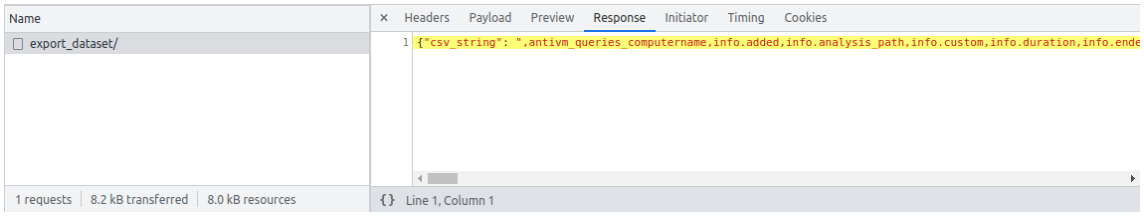
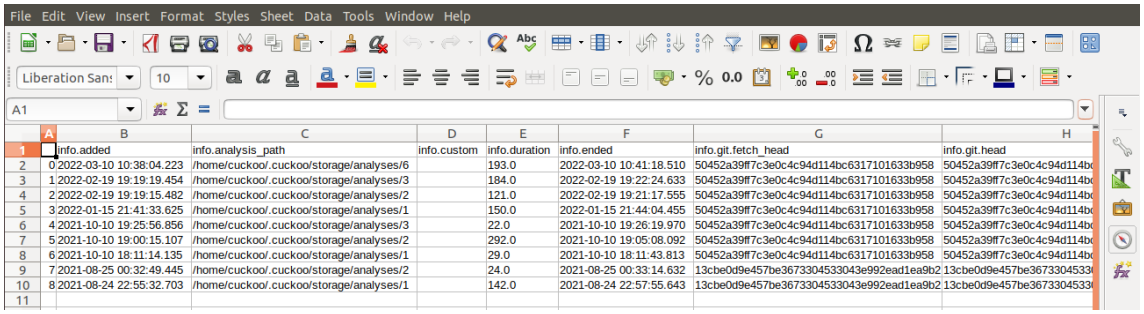


Figura 17. Resposta à requisição feita à *API* do *Cuckoo Web*.



	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	50452a39ff7c3e0c4c94d114b
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	50452a39ff7c3e0c4c94d114b
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	50452a39ff7c3e0c4c94d114b
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	50452a39ff7c3e0c4c94d114b
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	50452a39ff7c3e0c4c94d114b
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	50452a39ff7c3e0c4c94d114b
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	50452a39ff7c3e0c4c94d114b
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	13cbe0d9e457be3673304533
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	13cbe0d9e457be3673304533
11								

Figura 18. Exemplo de conjunto de dados, gerado no formato *CSV*.

de aprendizado de máquina na detecção de *malware*. Aqui se torna também necessária a criação de bons conjuntos de dados, capazes de subsidiar estudos específicos do domínio de cada tipo de *malware*, bem como uma constante atualização nessas bases de conhecimento, uma vez que novos ataques surgem com frequência.

Este trabalho busca contribuir com tais demandas, mitigando-as, à medida que facilita a criação de conjuntos de dados personalizados para diferentes domínios de pesquisa, sua reutilização e seu emprego no teste de diferentes modelos de aprendizado de máquina. A solução proposta estende ferramentas consagradas na análise de *malware*, fornecendo uma plataforma unificada que contém diversos códigos maliciosos já analisados e compartilhados, reduzindo o retrabalho na busca de tais arquivos, padronizando e melhorando a qualidade dos conjuntos de dados resultantes, além de servir como um arcabouço para futuros projetos (podendo ser adicionados métodos para evitar *sandbox evasion*, a criação de *Web crawlers* para continuamente povoar a base de dados, e a inclusão de novos dados de relatório à base, entre diversas outras possibilidades).

Referências

Borges, M., Labaki, A., Cattelan, R., and Miani, R. (2021). Construção de um conjunto de dados para análise estática de ransomwares. In *Anais Estendidos do XVII Simpósio Brasileiro de Sistemas de Informação*, pages 41–44.

Catak, F., Yazı, A., Elezaj, O., and Ahmed, J. (2020). Deep learning based Sequential model for malware analysis using Windows exe API Calls. *PeerJ Computer Science*, 6:e285.

Cuckoo Foundation (2010). What is cuckoo? https://cuckoo.readthedocs.io/en/latest/introduction/what. Acesso em: 16/05/2022.

Ehteshamifar, S., Barresi, A., Gross, T. R., and Pradel, M. (2019). Easy to fool? testing the anti-evasion capabilities of pdf malware scanners. *ArXiv*, abs/1901.05674.

Ferrand, O. (2015). How to detect the cuckoo sandbox and to strengthen it? *Journal of Computer Virology and Hacking Techniques*, 11:51–58.

Gibert, D., Mateu, C., and Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526.

Guibernau, F. (2020). Catch me if you can! - detecting sandbox evasion techniques. USENIX Association.

Miller, C., Glendowne, D., Cook, H., Thomas, D., Lanclos, C., and Pape, P. (2017). Insights gained from constructing a large scale dynamic analysis platform. *Digit. Investig.*, 22(S):S48–S56.

Sethi, K., Chaudhary, S. K., Tripathy, B. K., and Bera, P. (2018). A novel malware analysis framework for malware detection and classification using machine learning approach. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–4.

Yusirwan, S., Prayudi, Y., and Riadi, I. (2015). Implementation of malware analysis using static and dynamic analysis method. *Int. J. Comput. Appl.*, 117:11–15.