Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Classificação de requisições HTTP maliciosas por meio de aprendizagem de máquina

Diego Ignacio Zurita Rojas

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Profª. Dr. Daniel Macêdo Batista

São Paulo
2021

Agradecimentos

Agradecer a minha família, em especial a minha prima Pamela pelo apoio suporte incondicional, meus amigos, ao prof. Daniel Macedo Batista pela orientação e o prof. Vladimir Belitsky pelas conversas feitas durante o ano.

Resumo

Diego Ignacio Zurita Rojas. **Classificação de requisições HTTP maliciosas por meio de aprendizagem de máquina**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Com o grande volume de requisições a websites, o número de de requisições maliciosas que se aproveitam de vulnerabilidades aumenta propocionalmente. Nesse sentido, se faz necessária a automação de detecções de requisições maliciosas, já que a verificação manual nem sempre consegue ser rápida o suficiente para impedir a exploração das vulnerabilidades. Este trabalho estuda modelos de aprendizagem de máquina para tal automação por meio da análise de logs de servidor HTTP. Além disso, analisa seu desempenho em um Raspberry Pi.

Palavras-chave: Aprendizagem de máquina. Logs. Segurança Computacional. Redes de Computadores.

Abstract

Diego Ignacio Zurita Rojas. **Classification of malicious HTTP requests through machine learning**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

With the high volume of requests to websites, the number of malicious requests that take advantage of vulnerabilities increases proportionately. In this sense, it is necessary to automate the detection of malicious requests, as manual scanning is not always fast enough to prevent the exploitation vulnerabilities. This work studies machine learning models for such automation through the analysis of HTTP server logs. In addition, it analyzes performance on a Raspberry Pi.

Keywords: Machine learning. Logs. Computational Security. Computer Networks.

Lista de Abreviaturas

XSS	Cross Site Scripting
ETL	Extract Transform and Load
CSV	Comma Separated Value
CPU	Central Processing Unit
SVM	Suppor Vector Machine

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Com a variedade de serviços oferecidos pela Internet, a quantidade de requisições que são feitas a um servidor web aumenta. De acordo com Wiener e Bronson, 2014, o Facebook em 2014 recebia uma quantidade de consultas da ordem de bilhões por segundo. Nesse sentido é esperado que também aumente o número de requisições maliciosas, como por exemplo SQL injection, que está no top 10 da OWASP Andrew van der Stock e Gigler., 2017, uma organização sem fins lucrativos que visa melhorar a segurança em softwares, através da publicação de artigos, metodologias e documentação de maneira gratuita.

Em tal cenário se faz necessário um sistema automático de detecção de requisições maliciosas, pois analisar cada uma delas manualmente pode ser inviável ou bastante custoso. Uma possível solução é criar modelos de aprendizagem de máquina para encontrar padrões de requisições maliciosas e então tomar decisões automatizadas com base no resultado desses modelos.

Além disso, é desejável que esse projeto tenha um custo baixo em termos financeiros, de rede e de consumo de energia. Para isso, se tem proposto o uso de hardware de baixo custo para implantação desse tipo de sistema. Em L. Oshiro e Batista, 2019 há uma análise que conclui que é possível usar Raspberry Pi como nó de um cluster de processamento de dados, sendo que esses dados podem ser as requisições a um servidor HTTP.

1.1 Objetivos

O objetivo principal deste trabalho de conclusão de curso é construir modelos de aprendizagem de máquina para detectar requisições HTTP maliciosas. Tais modelos analisam logs de servidores HTTP para detectar potenciais ataques de SQL Injection e XSS, e devem ser executados em períodos fixos do dia, isto é, não está no escopo deste trabalho encontrar modelos que sejam executados em tempo real. Um segundo objetivo é, verificar o desempenho desses modelos quando executados em um Raspberry Pi.

Em relação a trabalhos anteriores, este TCC avança o que foi realizado no TCC "Análise de Desempenho de Computadores de Baixo Custo em um Sistema de Detecção de Intrusão" de Lucas Seiki Oshiro L. S. Oshiro, 2019.

1.2 Resultados alcançados

Este trabalho de conclusão de curso avaliou modelos de aprendizagem de máquina em dados simulados e reais, até concluir que o modelo de árvore de decisão é o melhor para o objetivo aqui estabelecido, chegando a uma acurácia próxima dos 93% em dados reais.

Além disso, comparou o desempenho nas tarefas de treinamento e classificação em um Macbook e um Raspberry Pi, e concluiu que o último pode ser usado nas tarefas de classificação desde que o tempo não seja um fator impeditivo, pois o tempo de classificação é, em média, 6 vezes maior em relação ao primeiro.

1.3 Organização da monografia

Os próximos capítulos desta monografia estão divididos da seguinte maneira:

- Capítulo 2: Trabalhos relacionados a este TCC e resultados encontrados.
- Capítulo 3: Conceitos e ferramentas que foram usados.
- Capítulo 4: Ambiente usado e como os dados foram coletados.
- Capítulo 5: Como o modelo final foi selecionado.
- Capítulo 6: Metodologia e os resultados dos experimentos.
- Capítulo 7: Conclusão.
- Capítulo 8: Avaliação pessoal e crítica.

Capítulo 2

Revisão da literatura

O uso de modelos de aprendizagem de máquina para detectar intrusões em servidores HTTP por meio de logs já foi estudado em outros trabalhos. Abaixo cito os artigos que nos serviram de referência e qual modelo foi utilizado.

- Baş Seyyar *et al.*, 2018: este trabalho analisa uma fase anterior dos ataques HTTP, a etapa de exploração de vulnerabilidades. Nessa etapa, em geral, um crawler pode ser executado em busca de possíveis vulnerabilidades, e nesse sentido o processo distinguir tais requisições das requisições de usuários ajuda na prevenção de ataques. Para isso, o trabalho propôs regras estabelecidas manualmente após uma análise dos logs, isto é, nenhum modelo de aprendizagem de máquina tradicional foi usado.
- Fontaine *et al.*, 2020: este trabalho visa detectar atividades maliciosas no ambiente da nuvem. Uma vez que sua adoção vem crescendo, a superfície de ataque cresce proporcionalmente. Nesse sentido, o trabalho propõem analisar arquivos de logs gerados pela nuvem e por aplicações web para treinar modelos de aprendizagem de máquina e automatizar a sua detecção. Para isso usou modelo de árvore de decisão e redes neurais.
- Joshi e Geetha, 2014: similar ao trabalho anterior, este trabalho propõe automatizar a detecção de ataques utilizando aprendizagem de máquina, contudo, neste trabalho o foco principal é o SQL injection motivado pela grave falta de privacidade que esta falha pode proporcionar. Para isso propõe um modelo de naive bayes.

Observar estes trabalhos direcionou a pesquisa no sentido de que os modelos de aprendizagem de máquina podem ser utilizados para detectar falhas de segurança em servidores HTTP, além disso qual modelo avaliar, e por fim quais dados foram utilizados.

Vale deixar registrado, um agradecimento ao autor Jaron Fontaine do trabalho Fontaine *et al.*, 2020, que gentilmente indicou quais logs reais foram usados em seu trabalho e onde encontrá-los.

Capítulo 3

Conceitos Básicos

Este capítulo apresenta os conceitos que foram utilizados para o desenvolvimento do TCC. Serão apresentadas as ferramentas utilizadas para realizar o Extract Transform and Load (ETL), simular os logs de ataques, modelos que foram utilizados e por fim quais frameworks foram usados para a aprendizagem de máquina.

3.1 Ferramentas

Kali Linux
: O Kali Linux é uma distribuição linux open source baseada em Debian. Tem como foco auxiliar em tarefas relacionadas a segurança da informação e testes de penetração. Neste trabalho seus utilitários foram utilizados para simular ataques.

xsser
: O xsser é um utilitário que está instalado por padrão no kali linux. Sua função é automatizar a detecção, exploração e análise de ataques XSS em aplicações web. Ele foi utilizado para gerar requisições HTTP maliciosas que exploram a falha XSS.

sqlmap
: O sqlmap é um utilitário que está instalado por padrão no kali linux. Sua função é automatizar a detecção e exploração de ataque de SQL injection em aplicações web. Ele foi utilizado para gerar requisições HTTP maliciosas que exploram a falha de SQL Injection.

Pandas
: O Pandas é uma ferramenta em python para o processo de análise de dados, especificamente neste trabalho, foi utilizado para realizar o ETL e análise dos logs coletados.

DVWA
: DVWA é uma aplicação web com falhas de segurança conhecidas que podem ser exploradas no ambiente local e controlado, facilitando assim a simulação de ataques. Aqui foi utilizado para simular um ambiente que tenha as falhas de segurança necessárias e com isso conseguimos gerar os logs necessários para o projeto.

3.2 Modelos de aprendizagem de máquina

Árvore de decisão

Árvore de decisão é um modelo de representação dos dados que visa classificar os dados tendo como base um conjunto de atributos.
Elas são construídas particionando os dados em conjuntos, chamados de nós, até que uma folha seja encontrada. Esta então é a classe do dados. Veja na Figura 3.1 em que é mostrada uma árvore de decisão que decide se um aluno está apto a se formar ou não. Nela, os quadrados são os nós e as elipses são as folhas, isto é, a respectiva classe do dado.

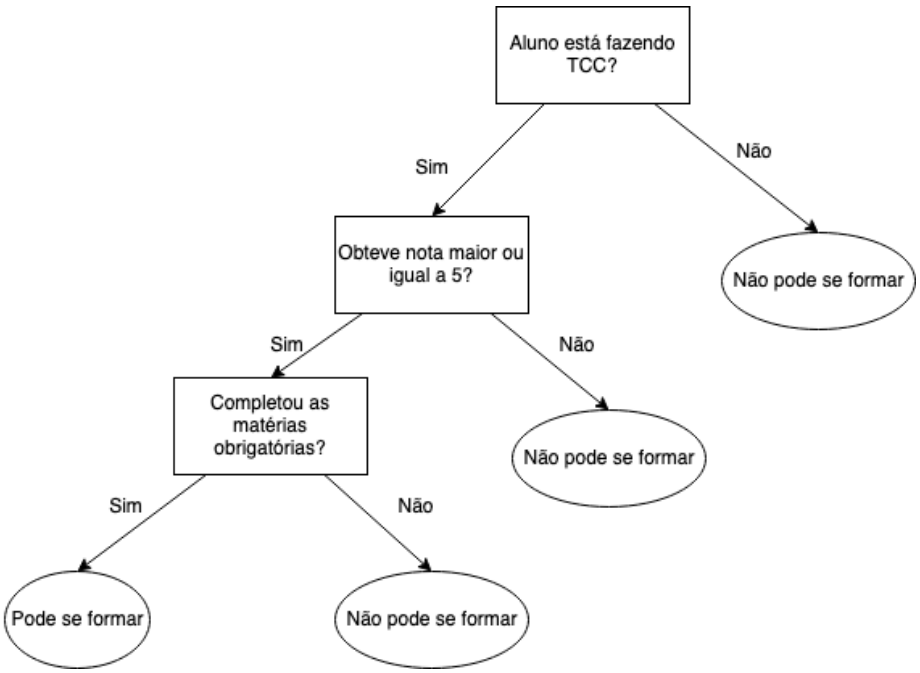

Figura 3.1: *Exemplo de árvore de decisão.*

Este modelo é útil pois a sua interpretabilidade é alta. A interpretabilidade é grau com que um modelo pode ser entendido em termos humanos.

Por exemplo, quando comparado com o modelo de florestas aleatórias que utiliza várias árvores de decisão para realizar a classificação, entender como cada árvore influencia na classificação final dificulta a interpretabilidade no contexto geral, portanto, o modelo de árvore de decisão ter um interpretabilidade maior do que o de florestas de decisão.

Florestas aleatórias

Florestas aleatórias é um modelo de aprendizagem de máquina que combina árvores de decisão. Isto é, uma certa quantidade de árvores são treinadas de maneira independente uma da outra, utilizando diferentes partes do conjunto de dados.
E para realizar a classificação, cada árvore realiza uma classificação (voto) e o resultado final da floresta é a classe com mais votos, como ilustrado na Figura 3.2 que apresenta uma floresta aleatória, onde a árvore 1 utiliza o conjunto de

características N_1 para realizar a classificação que resulta na classe C, a árvore 2 utiliza o conjunto de características N_2 para realizar a classificação D, e assim por diante. E a classe final é a classe C pois é a que possui a maior quantidade de votos:

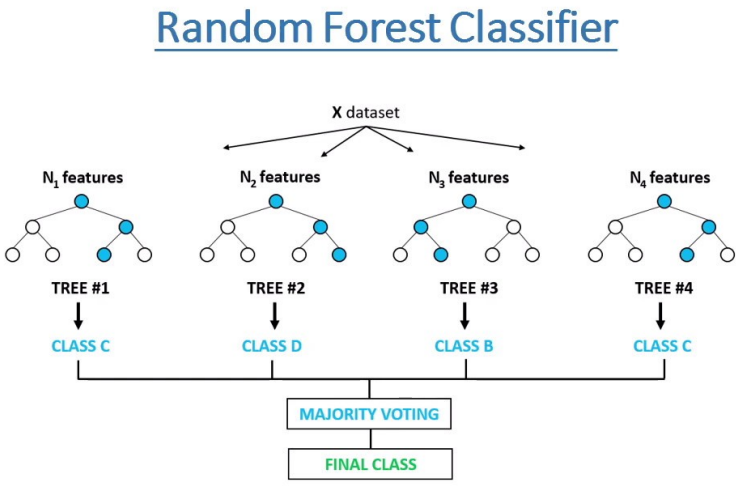

Figura 3.2: *Exemplo de floresta aleatória, retirado do trabalho* Habib *et al., 2020*

Este modelo possui dois pontos de atenção: sua interpretabilidade é baixa, dado o número de classificadores que temos que analisar, e também é consideravelmente mais pesado que o anterior, pois ele treina algumas árvores de decisão e não apenas uma.

3.3 Ferramentas de aprendizagem de máquina

scikit-learn

Scikit-learn é uma ferramenta que implementa algoritmos de aprendizagem de máquina, supervisionados e não supervisionados. Em particular, o modelo de árvore de decisão e floresta aleatória.
Além disso, ela integra com o ecossistema python, isto é: numpy, pandas e matplotlib.
E também fornece uma interface sólida, que facilita o seu uso.
Neste trabalho foi usado para treinar, avaliar e selecionar quais modelos foram utilizados para executar os experimentos.

Apache Spark

Apache Spark é uma ferramenta para trabalhar com aprendizagem de máquina de maneira distribuída ou em um único nó.
Assim como o scikit-learn, há alguns modelos implementados que estão prontos para serem utilizados, em particular, há implementações de árvore de decisão e floresta aleatória.
Há suporte para algumas linguagens, como: Python, Scala, Java e R.
Neste trabalho foi usado para executar os experimentos com o modelo selecionado.

3.4 Segurança em servidores HTTP

Ataque XSS

Ataque XSS consiste em scripts maliciosos que são injetados em websites confiáveis. Em geral, isso se dá por meio de formulários em websites.
Este ataque foi um objeto de estudo deste trabalho.

SQL Injection

SQL Injection consiste em enviar consultas modificadas a banco de dados por meio de entradas do usuário, com por exemplo campos de texto em formulários, as quais podem ler informação sensíveis do banco de dados as quais tal usuário não possui acesso.
Este ataque foi um objeto de estudo deste trabalho.

Registro de acessos nos logs

Nos registros de acesso de logs são armazenadas as informações de um requisição HTTP, neles podemos encontrar dados com o endereço IP de origem, data e hora e endereço requisitado, para citar alguns dados.
Neste trabalho, esses registros foram utilizados como fonte primária de dados para treinar e validar os modelos de aprendizagem de máquina.

3.5 Outros conceitos

Extract Transform and Load (ETL)

ETL é o processo de extrair, transformar e carregar dados de uma fonte de dados de origem para outro de destino. Nesse processo, características podem ser adicionadas, valores discrepantes removidos e padrões aplicados, para citar algumas operações comuns.
Neste trabalho foi utilizado para transformar os arquivos de logs do formato original para o formato tabular, que são usados nas ferramentas de aprendizagem de máquina.

Virtualização

Virtualização é o processo de executar múltiplas instâncias de sistemas operacionais em uma abstração do mesmo hardware. Com isso é possível criar instâncias isoladas de sistemas operacionais em um mesmo computador.
Neste trabalho, foi utilizado para criar as instâncias vítima e atacante.

Notebooks

Notebooks são arquivos divididos em células que executam trechos de código Python. Ele facilita a experimentação pois guarda os resultados das células, evitando assim a necessidade de executar o mesmo trecho de código múltiplas vezes.
Neste trabalho foi utilizado para realizar ETL, além de treinar e validar os modelos de aprendizagem de máquina.

Capítulo 4

Ambiente e dados

Este capítulo mostra o ambiente que foi usado para realizar as simulações de ataque, coletar os logs e treinar os modelos. Além disso, também será explicado como os logs foram coletados e processados.

Alguns dos itens foram inspirados no trabalho de conclusão de curso "Análise de Desempenho de Computadores de Baixo Custo em um Sistema de Detecção de Intrusão"de Lucas Seiki Oshiro L. S. Oshiro, 2019.

O ambiente aqui mostrado pode ser visto na Figura 4.1. Na máquina pessoal foram criadas duas máquinas virtuais para simular os ataques, o Google Colab usado para avaliar, treinar e compartilhar os modelos e por fim o Raspberry Pi e o Macbook para rodar os experimentos.

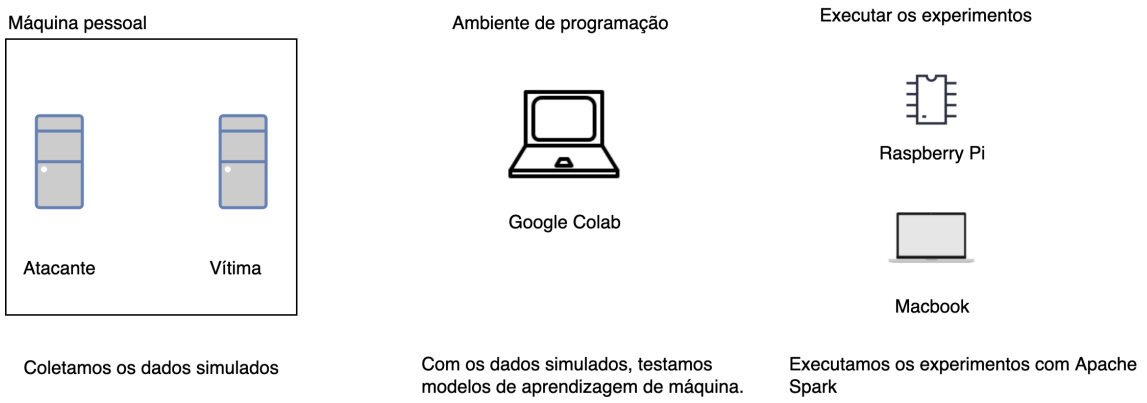

Figura 4.1: *Infraestrutura usada neste trabalho.*

4.1 Máquinas

4.1.1 Máquinas físicas

Neste trabalho, duas máquinas físicas foram utilizadas: um desktop e um notebook, além do ambiente Google Colab.

- Máquina pessoal
  - Sistema operacional: Ubuntu 20.04 LTS
  - Processador: Intel i7-10700, 8 núcleos, 4,8 GHz
  - Memória RAM: 32Gb

  Esta máquina foi utilizada para iniciar duas máquinas virtuais que serão usadas como o atacante e a vítima.
- Macbook
  - Sistema operacional: macOS Monterey - 12.0.1
  - Processador: M1, 8 núcleos, 3,2 GHz
  - Memória RAM: 8Gb

  Esta máquina foi utilizada para rodar os experimentos com Apache Spark, os quais serão comparados com os resultados dos computadores de placa única.

4.1.2 Máquinas virtuais

Aqui estão as duas máquinas virtuais que foram usadas: o atacante e a vítima, como mostrada na Figura 4.2. Elas foram criadas na máquina pessoal.

- Atacante
  - Sistema operacional: Kali Linux 2021.2
  - Processador: Intel i7-10700, 4 núcleos, 4,8 GHz
  - Memória RAM: 4Gb
- Vítima
  - Sistema operacional: Ubuntu 20.04 LTS
  - Processador: Intel i7-10700, 4 núcleos, 4,8 GHz
  - Memória RAM: 4Gb

4.1.3 Computador de placa única

Neste trabalho um Raspberry Pi foi utilizado, nele executamos as tarefas de treino e classificação dos logs.

- Raspberry Pi
  - Sistema operacional: Raspberry Pi OS - 10.9
  - Processador: Quad Core 1,2GHz Broadcom BCM2837
  - Memória RAM: 1Gb

4.2 Ambiente de programação

- Google Colab
  - Sistema operacional: Ubuntu 18.04.5 LTS
  - Processador: Intel(R) Xeon(R) CPU, 1 núcleo, 2,20GHz
  - Memória RAM: 16Gb

  Este ambiente foi utilizado para escrever os notebooks com os modelos e compartilhar os resultados parciais de forma eficaz. A configuração de hardware apresentada acima é a configuração padrão na data que este trabalho foi realizado.

4.3 Coleta de dados

A fonte primária de dados deste trabalho são logs de servidores web, especificamente, servidores HTTP Apache. Eles são úteis porque seguem um padrão, como mostrado abaixo:

```
127.0.0.1 - - [28/F...:28 -0400] "GET /url1 HTTP/1.0" 200 9 "-" "ApacheB"
127.0.0.1 - - [28/F...:28 -0400] "GET /url3 HTTP/1.0" 200 9 "-" "ApacheB"
127.0.0.1 - - [28/F...:28 -0400] "GET /url1 HTTP/1.0" 200 9 "-" "ApacheB"
127.0.0.1 - - [28/F...:28 -0400] "GET /url2 HTTP/1.0" 200 9 "-" "ApacheB"
127.0.0.1 - - [28/F...:28 -0400] "GET /url2 HTTP/1.0" 200 9 "-" "ApacheB"
```

Nele encontramos os seguintes dados: o endereço ip de acesso, a data, o método HTTP que foi utilizado, a URL de acesso, o código HTTP de retorno, a quantidade de bytes retornada e o agente que realizou a requisição.

Por exemplo, na última linha das sequências apresentadas acima, temos os seguintes dados:

- 127.0.0.1: endereço IP de origem, neste caso, o mesmo computador onde está a aplicação Apache.
- 28/F...:28 -0400: data e hora da requisição
- GET: é o verbo HTTP que foi utilizado. Ele significa que o usuário quer obter dados do servidor.
- /url2 HTTP/1.0: A URL que o usuário quer acessar e a versão do protocolo HTTP usado.
- 200: é código HTTP de resposta, neste caso, representa sucesso.
- 9: a quantidade de bytes retornados pela requisição.
- ApacheB: Representa o agente que realizou a requisição, aqui em específico ApacheB é uma abreviação para ApacheBenchmark o agente de um utilitário chamado ab [1], que é usado para automatizar um certa quantidade de requisições a uma certa URL.

[1] https://httpd.apache.org/docs/2.4/programs/ab.html

Nesse campo, em geral, estão as informações do navegador, como Google Chrome ou Firefox.

4.3.1 Dados simulados

No início do projeto não foi possível encontrar logs reais que fossem públicos. Algumas fontes consideradas foram o: Kaggle [2] e o IEEE Data Port [3]. Por causa disso, tomamos a decisão de simular requisições maliciosas e não maliciosas para então coletar seus logs. Para simular tais requisições três condições devem ser satisfeitas pois pensamos que seria prudente encontrar em logs reais, são elas:

- A quantidade de requisições não maliciosas deve ser consideravelmente superior a quantidade de requisições maliciosas. Isto é, os dados devem estar desbalanceados.
- Certas páginas devem ser mais acessadas do que outras, isso se dá por conta de que em sites há páginas que recebem mais acessos do que as outras, por exemplo, a página inicial.
- Deve haver uma predominância de acessos vindo dos navegadores Google Chrome e Firefox pelo fato deles estarem entre os navegadores mais usados no mundo [4].

Para alcançar tais condições, os utilitários xsser, sqlmap e um crawler desenvolvido para este trabalho foi usado (mais informações no apêndice). As requisições foram feitas da seguinte maneira:

- O utilitário xsser realiza requisições de ataques XSS.
- O utilitário sqlmap realiza requisições de SQL Injection.
- O crawler realiza as requisições não maliciosas

Os processos eram executados de modo que ao final, aproximadamente 10% das requisições fossem maliciosas e 90% não. Os acessos foram feitos a partir de máquinas separadas como mostrado na Figura 4.2.

Na simulação o fator tempo foi arbitrariamente desconsiderado. E nos modelos ele não se mostrou como uma característica relevante, mas isso não descarta a possibilidade dele ser usado com uma possível característica para classificar os dados.

[2] https://www.kaggle.com/datasets

[3] https://ieee-dataport.org/

[4] https://gs.statcounter.com/browser-market-share/

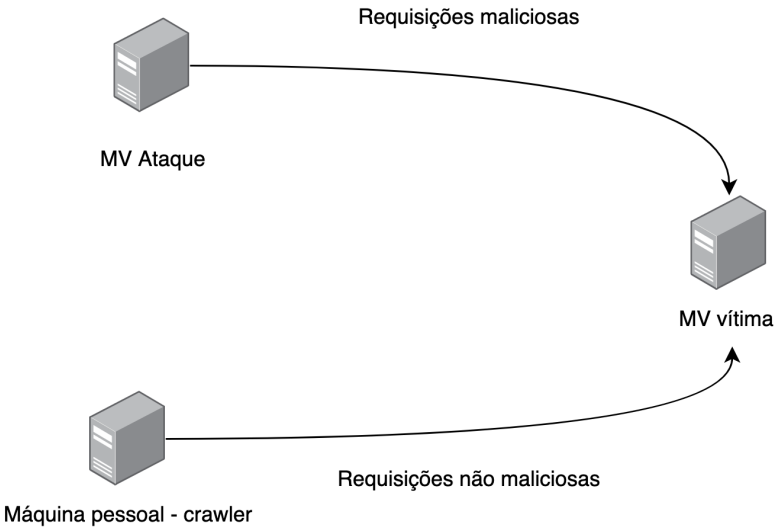

Figura 4.2: *Arquitetura para simular as requisições.*

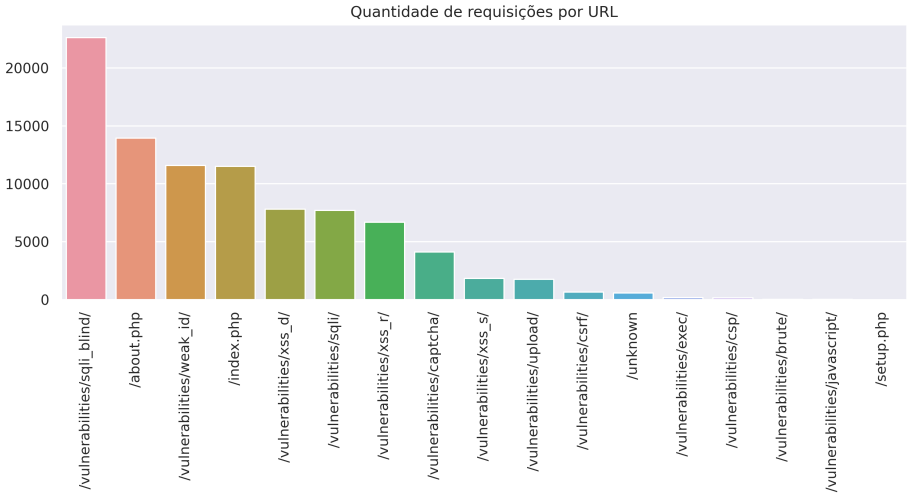

Figura 4.3: *Requisições por URL.*

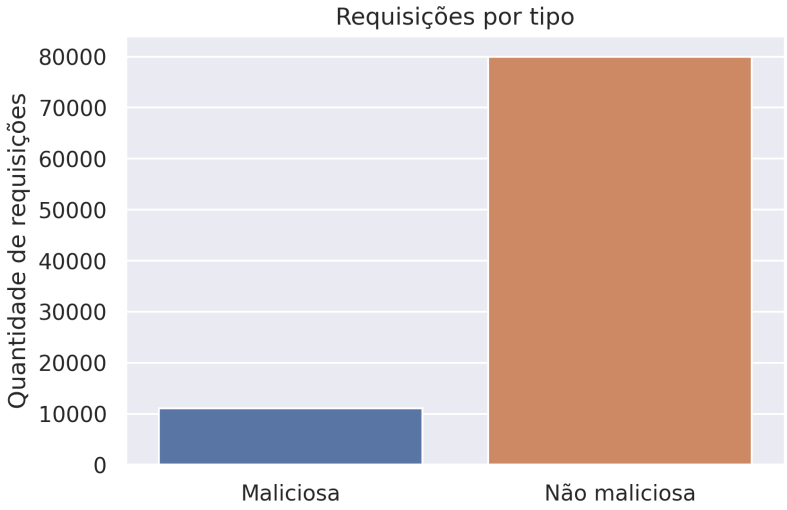

Figura 4.4: *Requisições por tipo.*

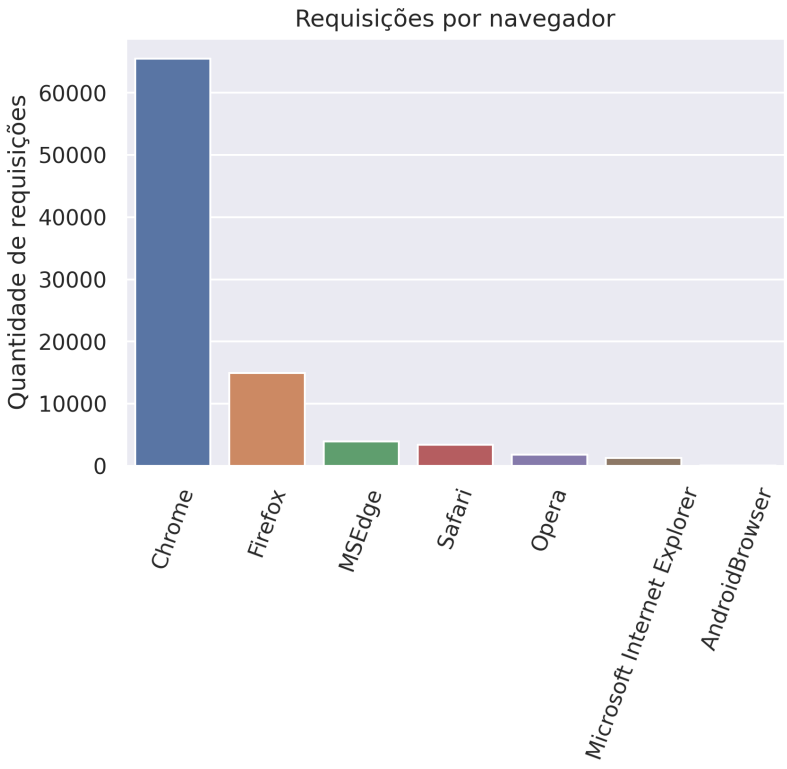

Figura 4.5: *Requisições por navegador.*

As figuras 4.3, 4.4 e 4.5 resumem as requisições realizadas. Nota-se que os critérios estabelecidos foram satisfeitos. Por exemplo, na Figura 4.3 há uma quantidade maior de requisições em certas URLs, na Figura 4.4 há mais requisições não maliciosas do que maliciosas e na Figura 4.5 vemos a predominância dos navegadores Google Chrome e Firefox. E por fim, como sabíamos a origem das requisições (xsser, sqlmap ou crawler) construir a fonte da verdade para estes dados foi uma tarefa trivial.

4.3.2 Dados reais

Em paralelo às simulações anteriores, procuramos logs de servidores reais para que os modelos encontrados fossem testados e validades neles.

Para isso entramos em contato com os autores dos artigos mencionados no Capítulo 2, e perguntamos quais dados foram usados em seus respectivos artigos. Jaron Fontaine, um dos autores do artigo Fontaine *et al.*, 2020 nos respondeu com os dados utilizados e eles estavam disponíveis de maneira pública.

Os dados indicados fazem parte de um projeto chamado The Honeynet Project [5]. Trata-se de uma organização de pesquisa sem fins lucrativos, que tem como objetivo pesquisar e investigar os últimos ataques e desenvolver ferramentas open source para melhorar a segurança da internet.

Os dados faziam parte de uma competição de segurança da informação, e foram coletados da seguintes maneira:

- Uma aplicação web com múltiplos servidores foi disponibilizada publicamente.
- Escolheu-se alguns dos servidores para subir uma versão da aplicação com falhas de segurança conhecidas.
- Então monitorou-se a atividade nesses servidores com falhas para detectar intrusões.

Apesar dos dados serem reais, a competição tratava-se de identificar, a partir dos logs, quais requisições eram maliciosas e também se o ataque foi realizado com sucesso ou não.

Nesse sentido, isso foi um desafio, dado que a fonte da verdade não era pública e a quantidade de logs coletados é consideravelmente grande. Então classificá-los manualmente foi necessário e para isso foi utilizado o script que se encontra no apêndice.

No fim, foram classificadas aproximadamente 10000 linhas e este conjunto de logs foi utilizado para validar os modelos em dados reais.

4.3.3 Estruturando os logs

Os logs em seu formato bruto não podem ser usados como entradas para modelos de aprendizagem de máquina, então um pré-processamento foi necessário. Por conta do padrão mencionado no início deste capítulo, foi possível criar um script que os estruturasse. Abaixo um trecho de código que foi usado nessa tarefa:

```
1    def transformLogFileToDataframe(logFilePath, isMalicious):
2        lineformat = re.compile(r"""(?P<ipaddress>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d
             {1,3}) - - \[(?P<dateandtime>\d{2}\/[a-z]{3}\/\d{4}:\d{2}:\d{2}:\d{2}
             (\+|\-)\d{4})\] ((\"(GET|POST) )(?P<url>.+)(http\/[1-2]\.[0-9]")) (?P<
             statuscode>\d{3}) (?P<bytessent>\d+) (?P<refferer>-|"([^"]+)") (["](?P
             <useragent>[^"]+)["])""", re.IGNORECASE)
3
4        logFile = open(logFilePath)
5
```

[5] https://www.honeynet.org/

```
6       logs_data = []
7
8       for l in logFile.readlines():
9          data = re.search(lineformat, l)
10
11         if data is None:
12            print("Falha ao fazer o parse da linha {} do arquivo {}. Pulando..".
                  format(l, logFile))
13            continue
14
15         datadict = data.groupdict()
16         datadict["malicious"] = isMalicious
17         logs_data.append(datadict)
18
19      logFile.close()
20
21      df = pd.DataFrame(logs_data)
22
23      df["statuscode"] = df["statuscode"].apply(int)
24      df["qtd_query_params"] = df["url"].apply(quantidade_de_params_na_query)
25
26      return df
```

No código acima, primeiro criamos uma expressão regular que irá capturar as informações de cada linha do log. Em seguida, abrimos o arquivo de log, aplicamos a expressão regular e em caso de sucesso, adicionamos ao dicionário logs_data que irá guardar as informações de cada linha. Ao final, temos o log em um formato tabular que pode ser usado nos modelos de aprendizagem de máquina.

Capítulo 5

Modelos de aprendizagem de máquina

Este capítulo tem como objetivo mostrar como os modelos foram escolhidos e apresentar métricas nos dados simulados e reais. Os resultados aqui apresentados foram obtidos no ambiente do Google Colab, uma vez que que este foi o ambiente escolhido para compartilhar os resultados dos nossos experimentos. Por fim, durante a seleção dos modelos apenas os dados simulados foram usados, uma vez que ainda não estávamos de posse dos logs reais, portanto apenas na última seção deste capítulo mostramos o desempenho do modelo escolhido nos dados reais.

5.1 Estrutura dos dados

No final do capítulo anterior mostrou-se como os arquivos brutos de logs foram transformados para um formato tabular. As colunas neles contidas podem ser vistas nas tabelas 5.1 e 5.2. A seguir é apresentado um dicionário do que significa cada coluna:

- ipaddress: o endereço IP do usuário que realizou a requisição.
- dateandtime: a data e hora em que a requisição foi realizada.
- url: a url requisitada.
- statuscode: o código HTTP de retorno.
- bytessent: quando bytes foram retornados na requisição.
- refferer: se a requisição se originou através de outro website, então o endereço deste website estará nesta coluna. Esse valor é passado através do cabeçalho Refere.
- useragent: o agente responsável pela requisição.
- malicious: variável indicadora que é 1 se a requisição é maliciosa, 0 caso contrário.

De posse dessas colunas que estão nos logs, mais 3 colunas foram criadas:

- url_sem_query: armazena a url requisitada sem o parâmetros de consulta.

ipaddress	dateandtime	navegador	statuscode	bytessent	refferer	malicious	qtd_query_params
192.168.0.219	13/May/2021:23:38:05 -0400	Safari	200	2937	-"	0	1
192.168.0.219	13/May/2021:23:38:05 -0400	Safari	200	2937	-"	0	0
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	1711	-"	0	0
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	1675	-"	0	1
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	1839	-"	0	2
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	1711	-"	0	1
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	2306	-"	0	0
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	1686	-"	0	0
192.168.0.219	13/May/2021:23:38:05 -0400	Chrome	200	1461	-"	0	1

Tabela 5.1: *Resultado final dos logs - parte 1.*

url_sem_query	url	useragent
/index.php	/index.php?extraParam=CwDK...	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3)..."
/index.php	/index.php	"Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_5; ar)..."
/vulnerabilities/sqli_blind/	/vulnerabilities/sqli_blind/	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1)..."
/vulnerabilities/sqli/	/vulnerabilities/sqli/?extraParam=DfoQ...	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3)..."
/vulnerabilities/xss_d/	/vulnerabilities/xss_d/?extraParam=QDIa...&extraParam2=bBj...	"Mozilla/5.0 (Windows NT 10.0; Win64; x64)..."
/vulnerabilities/sqli_blind/	/vulnerabilities/sqli_blind/?extraParam=Sodj...	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2)..."
/about.php	/about.php	"Mozilla/5.0 (Windows NT 6.2; WOW64)..."
/vulnerabilities/xss_r/	/vulnerabilities/xss_r/	"Mozilla/5.0 (Windows NT 5.1)..."
/vulnerabilities/weak_id/	/vulnerabilities/weak_id/?extraParam=isBe...	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3)..."

Tabela 5.2: *Resultado final dos logs - parte 2.*

- qtd_query_params: quantidade de parâmetros encontrados na query.
- navegador: armazena apenas o nome do navegador presente na coluna useragent.

5.2 Preparação dos dados

Durante o treino/validação de todos os modelos deste capítulo, o conjunto de dados original foi dividido em treino e teste, como explicado a seguir:

- treino: dados que serão usados para treinar o modelo.
- teste: dados não presentes no treino, com o objetivo de verificar se o modelo está generalizando bem.

Outro ponto de atenção necessário é manter o desbalanceamento da variável resposta nos conjuntos acima, dado que há mais requisições maliciosas do que não maliciosas.

Por fim, dados categóricos foram transformados em números usando a técnica de OneHotEncode. Este e o item anterior foram feitos com o auxílio dos utilitários de pré-processamento de dados do scikit-learn e do Apache Spark.

5.3 Modelo base

O modelo de florestas aleatórias foi escolhido como modelo base. Para avaliar o modelo, as colunas abaixo foram escolhidas de maneira arbitrária, já que neste ponto queríamos saber se modelos de árvores resolveriam nosso problema.

- statuscode
- navegador
- qtd_query_params
- bytessent

Com este modelo e as colunas mencionadas conseguimos uma acurácia de 1 nos dados de teste. Surpreendente, pois não esperávamos esse resultado na primeira tentativa, dado que se trata de um modelo base, logo suspeitamos que o modelo estava sobre ajustado nos dados.

Para ter uma intuição melhor de como o modelo estava realizando a classificação, optamos por experimentar o modelo de árvore de decisão, dado que depurá-lo é possível.

5.4 Depurando o modelo - árvore de decisão

As mesmas colunas da seção anterior foram usadas para treinar uma árvore de decisão, e novamente o modelo teve uma acurácia de 1 nos dados de treino e teste.

Para entender como a classificação está sendo realizada, exibimos a árvore final na Figura 5.1. Nesse sentido, como os dados passaram por um pré-processamento de OneHotEncoder vale mencionar o que significa cada índice em X:

- X[0] e X[1]: é a coluna statuscode.
- X[3] até X[9]: indicam o navegador que foi utilizado.
- X[10]: é a coluna qtd_query_params
- X[11]: é a coluna bytessent

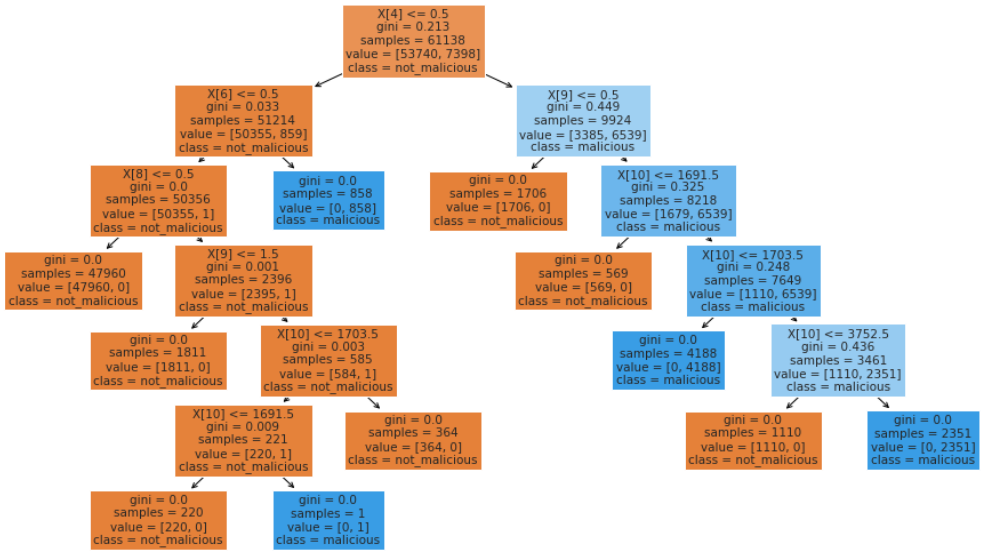

Figura 5.1: *Primeira árvore de decisão. Os nós laranjas representam a classe not_malicious e os azuis a classe malicious. Quanto mais escuro o nó, menor a probabilidade de erro na classificação.*

Vemos que a árvore está usando majoritariamente os dados da coluna navegador para realizar a classificação, o que não faz sentido, uma vez que esse dado foi simulado e o navegador não tem nenhuma relação com os ataques. Então removemos essa coluna da entrada e o resultado está na Figura 5.2.

Esta versão já nos pareceu mais sólida pois ela utiliza a quantidade de parâmetros na requisição e a quantidade de bytes enviados para o cliente, o que está relacionado com os ataques. Além disso, também atingiu uma acurácia de 1.

5.5 Escolha do modelo e considerações

Dado o que foi apresentado nas seções anteriores, somado aos resultados encontrados no capítulo 2, escolhemos o modelo de árvore de decisão com as colunas abaixo para realizar os experimentos e validá-lo em dados reais.

- statuscode
- qtd_query_params

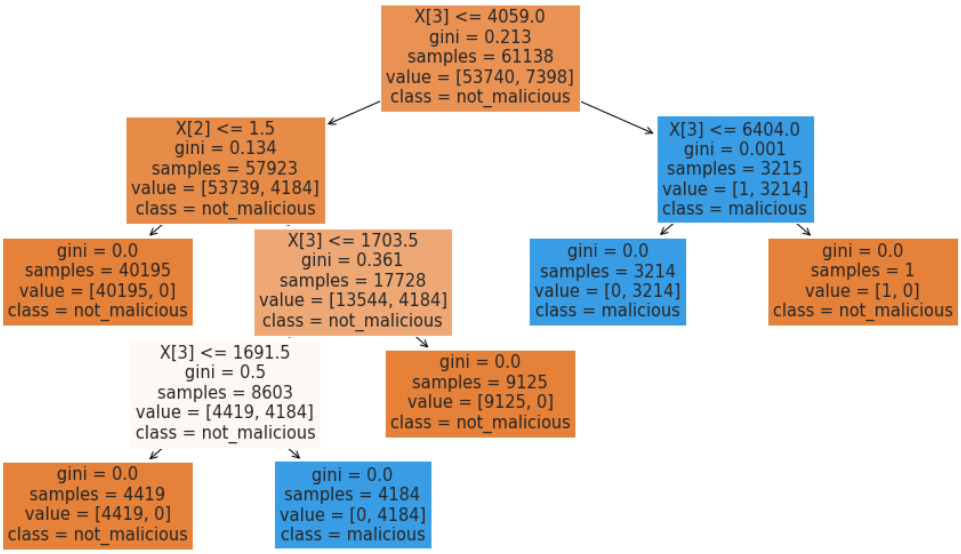

Figura 5.2: *Segunda árvore de decisão. Os nós laranjas representam a classe not_malicious e os azuis a classe malicious. Quanto mais escuro o nó, menor a probabilidade de erro na classificação.*

- bytessent

Outros modelos mais complexos, como redes neurais ou xgboost, não foram considerados pois o objetivo deste trabalho é usar o modelo em computadores onde os recursos, como memória e disco, são limitados portanto favorecendo modelos mais simples como o escolhido.

Também vale notar que características dos logs específicas a ataques XSS e SQL Injection, como verificar a existência de uma consulta SQL na url, não foram criadas pois as colunas padrões do log foram suficientes para conseguir uma acurácia considerada boa.

5.6 Testes em dados reais

De posse dos dados reais mencionados na Seção 4.3.2, executamos o modelo escolhido neles e atingimos uma acurácia de aproximadamente 0,93 no conjunto de teste. Portanto, definitivamente seguimos com ele nos experimentos.

Na Figura 5.3 vemos como a acurácia estabiliza em 0,93 a medida que mais dados são adicionados no conjunto de teste, ou seja, concluímos que o modelo está generalizando bem.

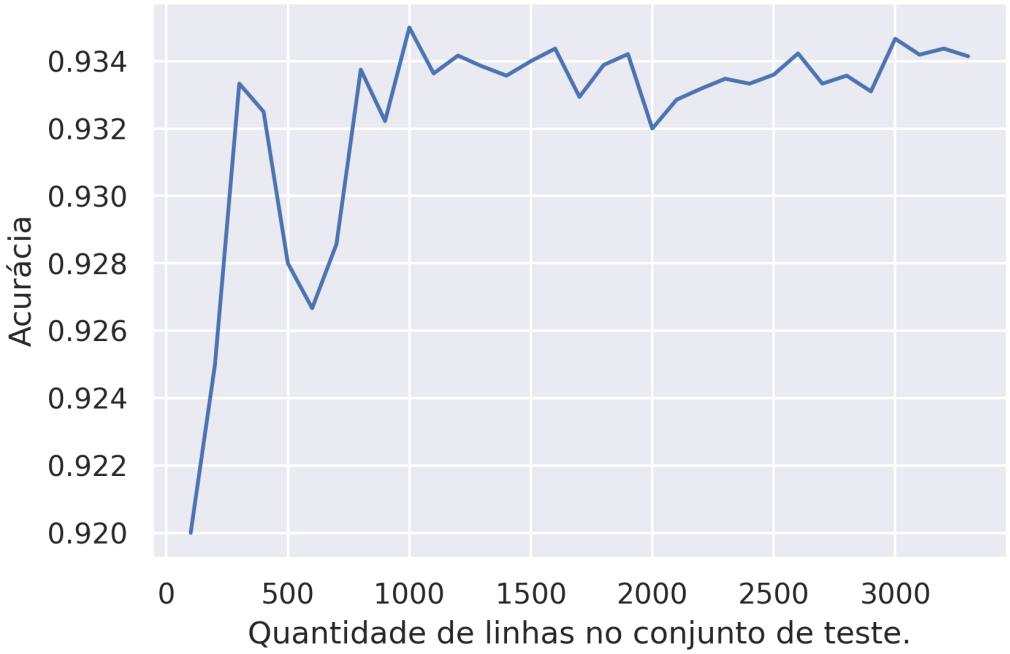

Figura 5.3: *A acurácia do modelo em dados reais pela quantidade de linhas*

Capítulo 6

Análise de desempenho

Este capítulo apresenta a metodologia e resultados dos experimentos, bem como expõe a sua análise.

6.1 Metodologia

Dois experimentos foram realizados: o primeiro para verificar a viabilidade do uso de Raspberry Pi para treinar modelos e o segundo usando-o somente para realizar as predições. Em ambos os casos, apenas o modelo de árvore de decisão foi usado.

Para o experimento de treino, a seguinte metodologia foi usada:

- Configuração do Apache Spark nos computadores.
- Início do script raspberry-log [1].
- Rodar 20 [2] vezes o script de treino e obter os seus tempos.
- Fim do script raspberry-log.
- Coletar as informações necessárias: tempo, consumo de memória e CPU.

Para avaliar o desempenho na tarefa de predições, três subconjuntos dos logs reais foram usados: com 1000, 5000 e 10000 linhas. E para isso a seguinte metodologia foi usada:

- Configuração do Apache Spark nos computadores.
- Início do script raspberry-log.
- Rodar 20 vezes o script de predição usando a entrada respectiva e obter os seus tempos.

[1] Este script foi desenvolvido no trabalho L. S. Oshiro, 2019. Ele captura dados de utilização de memória, disco e temperatura

[2] É necessário rodar mais de uma vez para obter dados mais confiáveis, evitando valores discrepantes. O número 20 foi escolhido de experimentos vistos e realizados nas disciplinas MAC0426 e MAC0219 do curso de ciência da computação do IME-USP.

- Fim do script raspberry-log.
- Coletar as informações necessárias: tempo, consumo de memória e CPU.

Os passos de ambos os casos foram executados no Macbook e no Raspberry Pi.

O script raspberry-log foi reaproveitado do trabalho L. S. Oshiro, 2019. Ele foi usado para coletar o consumo de memória e CPU. Para coletar o tempo, o utilitário time foi usado.

6.2 Experimentos

6.2.1 Controle

O objetivo deste experimento foi medir o consumo de memória e CPU dos computadores em repouso. Para isso o script raspberry-log foi executado por aproximadamente 10 horas, para termos valores mais confiáveis. Vale ressaltar que nenhum outro programa estava sendo executado, apenas o nosso script.

O macbook em repouso consumiu 5,4% de CPU com um desvio padrão de 0,6 e 679,34MB de memória com 30 de desvio padrão.

O Raspberry Pi em repouso consumiu 0,2% de CPU com um desvio padrão de 0,05 e 52,5MB de memória com 0,59 de desvio padrão.

Os valores acima foram considerados estáveis, portanto considerados como os valores do computador em repouso.

6.2.2 Treinamento

Aqui apresentamos os resultados do experimento onde foi usado o computador para treinar os modelos. Aqui um conjunto de dados de aproximadamente 7000 linhas foi usado para treinar e 3000 para validar o modelo.

O resumo dos valores coletados estão na Tabela 6.1. O desvio padrão está informado entre parênteses.

	Macbook	Raspberry Pi
CPU	22.58% (11.626)	41.42% (11.23)
RAM	1036.05MB (34.52)	270.94MB (82.28)
Tempo	18.54s (0.16)	110.96s (2.12)

Tabela 6.1: *Comparativo de métricas no experimento de treino.*

Note que o tempo médio de treinamento do Raspberry Pi é aproximadamente 10 vezes maior do que no Macbook, contudo isso não inviabiliza seu uso para treinar modelos, considerando que nesta etapa é natural esperar que treinar modelos em grande quantidades de dados leve um tempo considerável.

Observa-se também que o consumo de CPU do Raspberry Pi foi consideravelmente maior do que o observado em repouso, quando comparado ao Macbook. Nesse sentido, o

treino deve ser feito preferencialmente em um computador com configurações similares ao do Macbook utilizado.

6.2.3 Classificação

Este experimento realiza a classificação dos dados usando um modelo previamente treinado, da seguinte maneira:

- Treinamos o modelo previamente e salvamos no disco.
- Outro script carrega esse modelo na memória.
- Com o modelo na memória, a classificação é executada.

Os resultados desse experimento estão nas tabelas 6.2, 6.3 e 6.4:

	Macbook	Raspberry Pi
CPU	21.62% (12.01)	39.69% (11.80)
RAM	888.28MB (62.50)	250.46MB (77.83)
Tempo	15.93s (0.16)	84.87s (2.03)

Tabela 6.2: *Comparativo da classificação de 1000 linhas.*

	Macbook	Raspberry Pi
CPU	20.76% (10.36)	39.29% (11.46)
RAM	943.36MB (48.83)	256.45MB (78.85)
Tempo	16.30s (0.14)	91.72s (3.06)

Tabela 6.3: *Comparativo da classificação de 5000 linhas.*

	Macbook	Raspberry Pi
CPU	21.09% (10.74)	39.06% (11.78)
RAM	981.73MB (43.44)	257.01MB (80.69)
Tempo	16.53s (0.15)	93.22s (0.17)

Tabela 6.4: *Comparativo da classificação de 10000 linhas.*

Destas tabelas, observa-se que o tempo médio de execução do Raspberry Pi ainda é maior do que o do Macbook, classificando em média 107 logs/s e 604 logs/s, respectivamente. Isto é, o primeiro deve demorar, em média, 6 vezes mais para realizar classificações do que um computador tradicional com as configurações similares ao do Macbook utilizado, o que pode ser um impeditivo dependendo do volume de dados e da urgência em obter os resultados.

Além disso, era esperado que o experimento com 10000 linhas levasse pelo menos o dobro de tempo do que o com 5000 linhas, contudo isso não se observa principalmente pela paralelização aplicada pelo Apache Spark, diminuindo assim essa distância.

Capítulo 7

Conclusão

O modelo de árvore de decisão provou-se eficaz na tarefa de detectar intrusões a partir dos logs. Além disso, não foi necessária nenhuma característica específica dos ataques para atingir uma boa acurácia.

Para treinar o modelo, conclui-se que é preferível optar por um computador com especificações similares ao Macbook utilizado, dado o consumo de CPU que foi observado.

Por fim, para a classificação o Raspberry Pi pode ser utilizado quando o fator tempo não for um impeditivo, pois foi observado que ele demora em média 6 vezes mais para classificar do que o Macbook.

Capítulo 8

Avaliação pessoal e crítica

O tema deste trabalho foi escolhido principalmente por dois motivos:

- Aprofundar no tema de aprendizagem de máquina.
- Aprender mais sobre computação distribuída e Apache Spark.

O primeiro foi atingido, pois o contato e desenvolvimento nesse tema que eu tive foi muito produtivo e me ajudou em muito.

O segundo, foi atendido parcialmente, dado que o contato com Apache Spark ocorreu, contudo, durante os experimentos, o segundo Raspberry Pi queimou e a oportunidade de trabalhar com aprendizagem de máquina de maneira distribuída não ocorreu como eu esperava. Porém fica o aprendizado de estar preparado para qualquer adversidade em trabalhos futuros.

A crítica principal a este trabalho foi ter utilizado o Apache Spark em um único nó de Raspberry Pi, pois esse não é o objetivo da ferramenta, contudo o experimento e script já estavam prontos e alterá-los nos tomaria tempo que não dispúnhamos.

Por fim, durante o trabalho o conhecimento das seguintes disciplinas foi utilizado: sistemas operacionais para entender como utilizar da melhor maneira cada computador, ciência e engenharia de dados que mostrou ferramentas para processamento de dados, aprendizagem de máquina onde os modelos foram expostos de maneira mais formal, língua portuguesa para a escrita desta monografia e as disciplinas de estatística para entender melhor os modelos, além de ajudar a formalizar os experimentos. Por fim, a prática adquirida no grupo de extensão BeeData, um dos vários grupos de extensão do IME-USP, mostrou-se bastante útil neste trabalho.

Apêndice A

Crawler

O Crawler pode ser visto em https://github.com/DiegoZurita/MAC0499-TCC/blob/main/projeto/scripts/crawler.py. Ele foi utilizado para simular as requisições de acordo com o que foi explicado na seção 4.3.1.

O método execRequest realiza uma requisição com os parâmetros necessários para o DVWA. O método getRandomString gera uma palavra aleatória que será usada como parâmetros na requisição. E por fim no método createRequest, é gerado um conjunto de requisições tal que certas urls sejam mais requisitadas do que outras

Apêndice B

Classificador manual de log

O código fonte do classificador de logs pode ser encontrado em https://github.com/DiegoZurita/MAC0499-TCC/blob/main/projeto/scripts/log_classifier.py.

Nas primeiras linhas um arquivo de log é aberto. Na estrutura for o programa fica aguardando o usuário classificar cada linha como maliciosa e não maliciosa. Neste trabalho, foi utilizado para classificar as linhas de arquivos de logs reais.

Referências

[Andrew van der Stock e Gigler. 2017] Neil Smithline Andrew van der Stock Brian Glas e Torsten Gigler. *OWASP Top 10*. Out. de 2017. URL: https://owasp.org/www-project-top-ten/2017/Top_10.html (acesso em 07/03/2021) (citado na pg. 1).

[Baş Seyyar *et al.* 2018] Merve Baş Seyyar, Ferhat Özgür Çatak e Ensar Gül. "Detection of attack-targeted scans from the apache http server access logs". Em: *Applied Computing and Informatics* 14.1 (2018), pgs. 28–36. ISSN: 2210-8327. DOI: https://doi.org/10.1016/j.aci.2017.04.002. URL: https://www.sciencedirect.com/science/article/pii/S2210832717300169 (citado na pg. 3).

[Fontaine *et al.* 2020] Jaron Fontaine, Chris Kappler, Adnan Shahid e Eli De Poorter. "Log-based intrusion detection for cloud web applications using machine learning". Em: *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*. Ed. por Leonard Barolli, Peter Hellinckx e Juggapong Natwichai. Cham: Springer International Publishing, 2020, pgs. 197–210. ISBN: 978-3-030-33509-0 (citado nas pgs. 3, 15).

[Habib *et al.* 2020] Md Habib, Anup Majumder, Rabindra Nandi, Farruk Ahmed e Mohammad Uddin. "A comparative study of classifiers in the context of papaya disease recognition". Em: jan. de 2020, pgs. 417–429. ISBN: 978-981-13-7563-7. DOI: 10.1007/978-981-13-7564-4_36 (citado na pg. 7).

[Joshi e Geetha 2014] Anamika Joshi e V Geetha. "Sql injection detection using machine learning". Em: *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 2014, pgs. 1111–1115. DOI: 10.1109/ICCICCT.2014.6993127 (citado na pg. 3).

[L. Oshiro e Batista 2019] Lucas Oshiro e Daniel Batista. "Análise preliminar da detecção de ataques ofuscados e do uso de hardware de baixo custo em um sistema para detecção de ameaças". Em: *Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Gramado: SBC, 2019, pgs. 233–240. DOI: 10.5753/sbrc_estendido.2019.7792. URL: https://sol.sbc.org.br/index.php/sbrc_estendido/article/view/7792 (citado na pg. 1).

[L. S. Oshiro 2019] Lucas Seiki Oshiro. *Análise de Desempenho de Computadores de Baixo Custo em um Sistema de Detecção de Intrusão*. Dez. de 2019. URL: https://owasp.org/www-project-top-ten/2017/Top_10.html (acesso em 07/03/2021) (citado nas pgs. 1, 9, 25, 26).

[Wiener e Bronson 2014] Janet Wiener e Nathan Bronson. *Facebook's Top Open Data Problems*. Out. de 2014. URL: https://research.fb.com/blog/2014/10/facebook-s-top-open-data-problems/ (acesso em 07/03/2021) (citado na pg. 1).