UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Levi da Silva Ramos Júnior

REQUESTBERT-BILSTM:
Detecção de ataques em Requisições HTTP sem Log Parser

Recife

2022

Levi da Silva Ramos Júnior

REQUESTBERT-BILSTM:
Detecção de ataques em Requisições HTTP sem Log Parser

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação

Área de Concentração: Inteligência computacional

Orientador: Cleber Zanchettin

Recife

2022

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

R175r Ramos Júnior, Levi da Silva
RequestBERT-BiLSTM: detecção de ataques em requisições HTTP sem log parser / Levi da Silva Ramos Júnior. – 2022.
91 f.:il., fig, tab.

Orientador: Cleber Zanchettin.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.

Inclui referências.

1. Inteligência computacional. 2. Detecção de ataques. I. Zanchettin, Cleber (orientador). II. Título.

006.31 CDD (23. ed.) UFPE - CCEN 2022-197

Levi da Silva Ramos Junior

“REQUESTBERT-BiLSTM:
Detecção de ataques em Requisições HTTP sem Log Parser”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovado em: 30/11/2022.

BANCA EXAMINADORA

Prof. Dr. Divanilson Rodrigo de Sousa Campelo,
Centro de Informática / UFPE

Prof. Dr. Bruno José Torres Fernandes
Escola Politécnica / UPE

Prof. Dr. Cleber Zanchettin
Centro de Informática / UFPE
(**Orientador**)

Dedico esse trabalho à minha esposa e ao meu filho, que contribuíram cada um com à sua maneira para que esse trabalho fosse concluído.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por ter me proporcionado essa oportunidade em realizar um mestrado, algo que achava muito distante de mim por diversos motivos. Agradeço também à minha família: minha esposa, Camila, e ao meu filho, Leonardo. Minha esposa que sempre me incentivava a tentar o mestrado, e quando passei, me incentivou a não desistir e a concluí-lo. Logo, só tenho que a agradecê-la por toda sua compreensão e apoio. E quanto ao meu filho, Léo, que por muitas vezes me fez estudar com ele no colo, só tenho que agradecê-lo por me fazer superar os meus limites e me tornar um pai cada dia melhor. E por fim, aos meus pais e meu irmão que, mesmo longe, sempre acompanhavam e ficavam curiosos em cada etapa durante esse tempo no mestrado.

RESUMO

No cenário atual da internet, a maioria dos serviços, como compartilhamento de informações, entretenimento e educação, são prestados por servidores Web. Com o surgimento de diversos serviços, a Web se tornou o principal local de atuação para invasores e fraudadores. A maioria das técnicas defensivas nos servidores Web não consegue lidar com a complexidade e evolução dos ataques cibernéticos em requisições HTTP. No entanto, as abordagens de aprendizagem de máquina podem ajudar a detectar ataques, sejam eles conhecidos ou desconhecidos. Neste trabalho, será apresentado o modelo *RequestBERT-BiLSTM*, o qual permite detectar ataques em requisições HTTP sem a utilização de *Log Parser* ou analisador de log. Para aferir o desempenho do modelo proposto foi necessário testá-lo nos conjuntos de dados públicos: *CSIC 2010, ECML/PKDD 2007, BGL* e no conjunto de dados construído neste trabalho baseado em um ambiente real, onde as requisições foram extraídas do ativo de segurança: *F5 Big-IP*. Observou-se que o modelo proposto teve desempenho superior aos modelos criados e da literatura. Outra contribuição deste trabalho é a dificuldade que a etapa de análise de log pode trazer devido a erros gerados pelos métodos tradicionais de analisadores de logs. Os experimentos realizados com os analisadores de log demonstram a dificuldade que esse processo traz ao problema de detecção de ataques. A proposta ainda sugere que modelos baseados em aprendizado de máquina são estratégias promissoras para detecção de ataques na Web.

Palavras-chaves: requisição HTTP; requestbert-bilstm; detecção de ataques; análise de log.

ABSTRACT

In the current internet scenario, most services such as information sharing, entertainment and education are provided by web servers. With the emergence of various services, the Web has become the main place of action for attackers and fraudsters. Most defensive techniques on web servers cannot handle the complexity and evolution of cyber attacks on HTTP requests. However, machine learning approaches can help detect attacks, whether known or unknown. In this work, the *RequestBERT-BiLSTM* model will be presented, which allows detecting attacks in HTTP requests without using *Log Parser* or a log analyzer. To assess the performance of the proposed model, it was necessary to test it on public datasets: *CSIC 2010, ECML/PKDD 2007, BGL* and on the dataset built in this work based on a real environment, where the requests were extracted security asset: *F5 Big-IP*. It was observed that the proposed model performed better than the models created and in the literature. Another contribution of this work is the difficulty that the log analysis step can bring due to errors generated by traditional methods of log analyzers. The experiments carried out with the log analyzers demonstrate the difficulty that this process brings to the attack detection problem. The proposal also suggests that models based on machine learning are promising strategies for detecting attacks on the Web.

Keywords: HTTP requests; requestbert-bilstm; attack detection; log parser.

LISTA DE FIGURAS

Figura 1 – Tipos de ataques ocorridos durante a segunda metade do ano de 2021. . . 15
Figura 2 – Matriz Enterprise MITRE *OWASP*. 22
Figura 3 – Lista dos 10 maiores riscos *OWASP*. 23
Figura 4 – Ataque de quebra de controle de acesso. 24
Figura 5 – Ataque de falha de criptografia. 26
Figura 6 – Ataque de Injeção. 27
Figura 7 – Ataque de Design Inseguro. 28
Figura 8 – Ataque de configuração incorreta de segurança. 29
Figura 9 – Linha de Requisição 31
Figura 10 – Componentes da Requisição HTTP. 31
Figura 11 – Componentes da Requisição HTTP. 33
Figura 12 – Exemplo do resultado do *Log Parser* em uma requisição HTTP. 35
Figura 13 – Analisadores de log em suas categorias. 36
Figura 14 – Arquitetura *LSTM* 41
Figura 15 – Arquitetura BiLSTM 42
Figura 16 – Representação Word Embeding 44
Figura 17 – Arquitetura Transformers 45
Figura 18 – Etapas do *BERT*. 47
Figura 19 – Exemplo de uma Query String. 54
Figura 20 – Arquitetura do RequestBERT-BiLSTM. 58
Figura 21 – Log original capturado no ativo de segurança. 59
Figura 22 – Apenas o campo *Request* foi retirado do log original conforme a Figura 21. Uma requisição HTTP de um possível ataque XSS ao servidor web. 60
Figura 23 – Requisição HTTP após o tratamento inicial: colocando em minúsculas e retirando os caracteres especiais. 60
Figura 24 – Divisão da palavra em várias subpalavras, após a tokenização. 60
Figura 25 – IDs exclusivos das palavras ou subpalavras. 60
Figura 26 – Preenchimento com padding [PAD]. 60
Figura 27 – Tensor binário que indica a posição dos índices preenchidos com [MASK]. 60

Figura 28 – Gráfico mostrando o comportamento do tamanho das requisições do Dataset Log-Security. 62
Figura 29 – Requisição HTTP extraída do log original de um possível ataque. 67
Figura 30 – Template criado pelo analisador - **DRAIN** 67
Figura 31 – Template criado pelo analisador - **AEL** 67
Figura 32 – Template criado pelo analisador - **LogSig** 67
Figura 33 – Template criado pelo analisador - **LFA** 67
Figura 34 – Sistema Big-IP. 70
Figura 35 – UMAP do dataset Log Security. 72
Figura 36 – Distribuição das classes do dataset Log Security. 73
Figura 37 – Gráfico com os desempenho dos métodos de Log Parser. 77
Figura 38 – Tempo na predição dos modelos analisados. 81
Figura 39 – Desempenhos dos modelos. Os modelos com *BERT* sempre superiores aos outros e o RequestBERT-BiLSTM sempre superior a todos. Vale ressaltar que, os modelos Word2Vec, *CNN* e NeuralLog tiveram o problema do paradoxo de acurácia. 81
Figura 40 – Comparação dos modelos experimentados com e sem *Log Parser*. 82
Figura 41 – Exemplo de uma entrada do Log do Dataset BGL e logo abaixo mostra os campos fixos da estrutura do Log. 82

LISTA DE TABELAS

Tabela 1 – Tabela dos Datasets 70
Tabela 2 – Tabela de comparação dos Analisadores de Logs no dataset Log-Security . 76
Tabela 3 – Tabela de comparação dos analisadores de logs na base de dados, Log-Security. 77
Tabela 4 – Comparação do Desempenho entre os Modelos e os Datasets 80

SUMÁRIO

1 INTRODUÇÃO 14
1.1 MOTIVAÇÃO 17
1.2 OBJETIVOS 19
1.3 OBJETIVOS ESPECÍFICOS 19
1.4 APRESENTAÇÃO 20
2 DETECÇÃO DE ATAQUES 21
2.1 MITRE ATT&CK 22
2.2 OWASP TOP 10 VULNERABILIDADES 23
2.2.1 *Broken Access Control* 24
2.2.2 *Cryptographic Failures* 25
2.2.3 *Injection Attack* 25
2.2.4 *Insecure Design* 27
2.2.5 *Security misconfiguration* 28
2.3 CONSIDERAÇÕES 29
3 REVISÃO DA LITERATURA 30
3.1 REQUISIÇÕES HTTP 30
3.1.1 Linha de Requisição 30
3.1.2 Cabeçalhos HTTP 31
3.1.3 Corpo de mensagem 32
3.2 DETECÇÃO DE ANOMALIAS 33
3.3 LOG PREPROCESSING 34
3.3.1 *Log Parsing* ou Análise de Log 35
3.3.1.1 Drain 37
3.3.1.2 AEL 37
3.3.1.3 LogSig 38
3.3.1.4 LFA 39
3.4 APRENDIZAGEM PROFUNDA 39
3.4.1 LSTM 40
3.4.2 BiLSTM 41
3.5 PROCESSAMENTO DE LINGUAGEM NATURAL- PLN 42

3.5.1 Representação Textual . . . 43
3.6 TRANSFORMERS . . . 44
3.6.1 Bidirectional Encoder Representations from Transformers (BERT) . 46
3.6.1.1 Masked LM . . . 47
3.6.1.2 Next Sentence Prediction (NSP) . . . 48
3.7 TÉCNICAS E MODELOS COMPARATIVOS . . . 48
3.7.1 Word2Vec . . . 48
3.7.2 NeuralLog . . . 49
3.7.3 CNN . . . 49
3.7.4 Modelos BERT . . . 49
3.7.5 Considerações . . . 50
4 TRABALHOS RELACIONADOS . . . 51
4.1 DETECÇÃO DE ATAQUES EM REQUISIÇÕES HTTP . . . 51
4.2 DETECÇÃO DE ATAQUES BASEADAS EM URL . . . 52
4.3 DETECÇÃO DE ANOMALIAS EM OUTROS TIPOS DE LOGS . . . 54
4.4 CONSIDERAÇÕES . . . 56
5 REQUESTBERT-BILSTM . . . 57
5.1 PRÉ-PROCESSAMENTO . . . 59
5.2 TOKENIZAÇÃO . . . 61
5.3 TRANSFORMER-BASED CLASSIFICATION . . . 62
5.4 DETECÇÃO DE ATAQUES . . . 63
6 EXPERIMENTOS . . . 64
6.1 MODELOS AVALIADOS . . . 64
6.2 MÉTODOS DE ANALISADORES DE LOGS . . . 65
6.3 DATASETS . . . 68
6.3.1 Dataset - Log Security . . . 70
6.4 MÉTRICAS UTILIZADAS . . . 73
6.4.1 Precisão . . . 74
6.4.2 *Recall* . . . 74
6.4.3 *F1-Score* . . . 74
6.4.4 Acurácia . . . 75
6.4.5 ROC e AUC . . . 75
7 RESULTADOS E DISCUSSÃO . . . 76

7.1 COMPARATIVOS ENTRE OS MÉTODOS DE LOG PARSER 76
7.2 COMPARATIVOS ENTRE OS MODELOS ANALISADOS 77
7.2.1 RequestBERT-BiLSTM x Word2Vec 77
7.2.2 RequestBERT-BiLSTM x CNN 78
7.2.3 RequestBERT-BiLSTM x NeuralLog 78
7.2.4 RequestBERT-BiLSTM x BERT 79
7.2.5 RequestBERT-BiLSTM x BERT-CNN 79
7.3 COMPARAÇÃO DE RESULTADOS 79
8 CONCLUSÃO 84
8.1 CONTRIBUIÇÕES 85
8.2 TRABALHOS FUTUROS 85
REFERÊNCIAS 87

1 INTRODUÇÃO

Servidores Web são amplamente utilizados em várias organizações e são alvos constantes de inúmeros ataques que podem causar grandes danos. Para reduzir o risco de ataques a servidores Web, os desenvolvedores e especialistas em segurança precisam criar sistemas seguros.

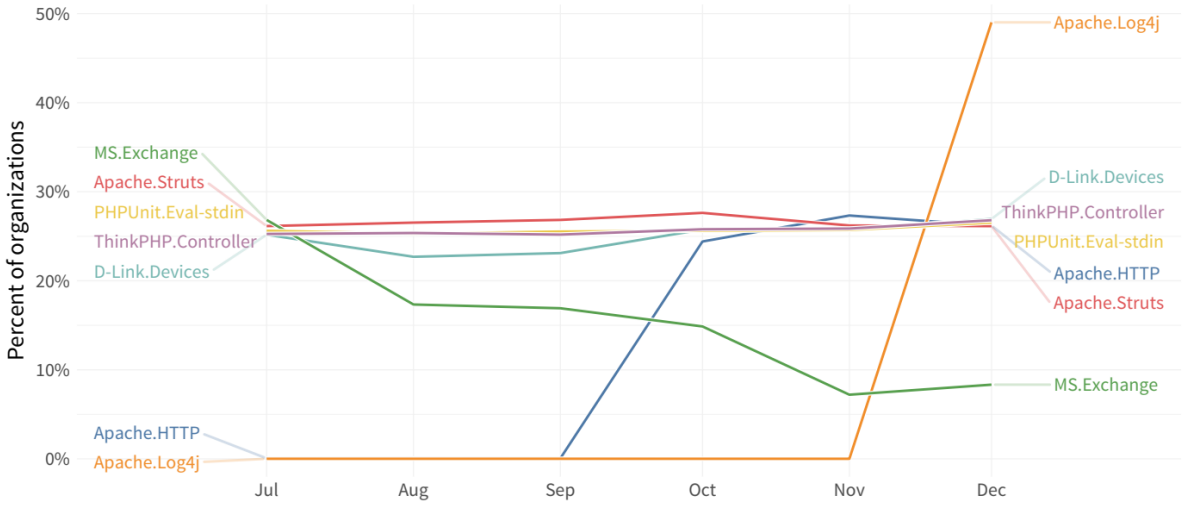
A comunicação existente entre os clientes e os servidores Web é basicamente realizada por mensagens HTTP. Há dois tipos de mensagens: requisições (*requests*) enviadas pelo cliente para disparar uma ação no servidor, e respostas (*responses*), a réplica do servidor. Essas mensagens HTTP são compostas de informação textual codificada em *ASCII*, e se espalham por múltiplas linhas. Em HTTP/1.1, e versões anteriores do protocolo, estas mensagens eram abertamente enviadas através da conexão. Já em HTTP/2, a mensagem antes legível por humanos é agora dividida em quadros HTTP, resultando em otimização e evolução de desempenho.(MDN, 2021)

As requisições HTTP estão presentes em qualquer cenário que ocorra a comunicação entre clientes e servidores. Neste contexto, as requisições podem ser legítimas ou maliciosas. Requisições legítimas são aquelas que o cliente requer um serviço provido por um servidor sem objetivo fraudulento, já a requisição maliciosa trata-se de uma atividade com fins suspeitos ou ilegais com objetivo de extrair, ou expor informações do servidor.

Estudos apontam que em 2021 houve 50% a mais de ataques por semana nas redes corporativas em comparação a 2020. O gráfico da Figura 1 mostra o aumento dos ataques para o ano de 2021. Além disso, é seguro afirmar que grande parte dos ataques que ocorrem no mundo são direcionados a serviços na Web (CISCO, 2022; FORTINET, 2022).

Grandes empresas no domínio de Segurança em Tecnologia da Informação, criam diariamente milhares de formas para tentar evitar qualquer tipo de ataque. No entanto, há uma grande dificuldade dessas empresas em evitar os ataques, tendo em vista que eles são construídos baseados nas próprias proteções existentes dos servidores, ou seja, a partir das tecnologias que fornecem o serviço de proteção é que surgem os ataques com as características correlacionadas àquela tecnologia. Diante deste cenário, existem inúmeras soluções de proteção de segurança Web, dentre elas há o *Firewall* de Aplicação Web, do inglês, *Web Aplicattion Firewall (WAF)*. O *WAF* é um tipo de *Firewall* que se aplica especificamente a serviços Web. Ao inspecionar o tráfego HTTP, ele pode impedir ataques originados da internet a partir de falhas de segurança nos servidores. No entanto, os *WAFs* atuais funcionam normalmente baseado em regras e dependem muito de assinaturas para detectar e prevenir ataques. Além do que, eles necessitam

Figura 1 – Os tipos de ataques ocorridos durante a segunda metade do ano de 2021 com relação ao percentual das empresas protegidas pelos ativos FORTINET.



Fonte: (FORTIGATE, 2022)

de capacidade suficiente para caracterizar e generalizar visando cobrir comportamentos normais ou maliciosos, considerando que, na prática, é uma tarefa prolongada e custosa atualizar as regras contra novos ataques.

O ativo de segurança utilizado neste trabalho foi o *F5 BIG-IP* que além de ser um *WAF* ele foi projetado para disponibilidade de aplicativos, controle de acesso e soluções de segurança para servidores Web.

Os servidores Web geralmente são alvos fáceis de ataques cibernéticos porque podem ser acessados pela rede. Monitorar aplicativos Web para ataques e alertar os usuários quando o ataque é detectado é o trabalho de um sistema de detecção de intrusão. A filtragem do tráfego da Web é uma tarefa altamente complexa e desafiadora devido à sua natureza dinâmica. Existem vários ataques disponíveis, como *Man in the Middle, DDOS, Malware, Email Phishingattack*, etc. Este trabalho se restringe nos ataques Web baseados em requisições HTTP e rapidamente nota-se que há muitos desafios a serem superados, incluindo a detecção de ataques desconhecidos, detecção falsos positivos, identificação de consultas incertas e detecção de comportamentos anormais.

Identificar ameaças pode ser difícil e está sujeito a erros. O uso de uma estrutura de modelagem fornece estrutura para o processo de modelagem de ameaças e pode incluir outros benefícios, como estratégias de detecção e contramedidas. Então neste cenário exitem algumas estruturas bem conhecidas: *OWASP top 10* e *MITRE ATT&CK Framework*.

A lista *OWASP top 10* é um dos produtos mais famosos do *Open Web Application Security Project (OWASP)*. Seu foco é encontrar as principais vulnerabilidades de aplicativos Web. Esta famosa lista é atualizada com as vulnerabilidades mais comuns ou perigosas detectadas em aplicativos Web. Já o *MITRE ATT&CK* foi projetado para oferecer suporte à segurança cibernética, fornecendo uma estrutura para modelagem de ameaças, testes de penetração, desenvolvimento de defesa e exercícios de segurança cibernética. O *MITRE ATT&CK* divide o ciclo de vida de um ataque cibernético em quatorze estágios chamados de "Táticas" pelo *MITRE*. Cada uma dessas táticas descreve uma meta específica que um invasor pode precisar alcançar em seu caminho para atingir seu objetivo geral, como escalar privilégios ou obter acesso às credenciais da conta. (POSTON, 2021)

A lista *OWASP top 10* é um excelente ponto de partida ao realizar um exercício de modelagem de ameaças para aplicativos Web. No entanto, embora a lista *OWASP top 10* esteja focada em vulnerabilidades de aplicativos Web, isso não significa que ela tenha valor apenas para desenvolvedores Web.

Outra forma encontrada de combater os ataques a servidores Web é com o uso de inteligência artificial, cujo objetivo é identificar padrões em ataques e realizar um processo de classificação, enquadrando o ataque ao tipo adequado, isso através da utilização de *Natural Language Processing(NLP)* ou Processamento de Linguagem Natural (PLN). O NLP oferece a capacidade de um computador entender, analisar, manipular e potencialmente gerar a linguagem humana, e como as requisições HTTP não são textos cursivos, mas carregam informações características, o NLP é essencial para esta atividade. Nesse contexto, os logs, são arquivos que registram informações detalhadas sobre eventos gerados por computadores que acessam ou tentam acessar.

A detecção de anomalias é um tópico importante discutido em várias áreas de pesquisa e domínios de aplicação. Anomalias são definidas como instâncias de dados que se destacam como diferindo de todos os outros(CHALAPATHY; CHAWLA, 2019). A detecção de anomalias indica o problema de descobrir padrões que não cumprem os comportamentos esperados (CHANDOLA; BANERJEE; KUMAR, 2009). Por exemplo, em um servidor Web, a detecção de anomalias pode monitorar chamadas de sistema, ocorrência de eventos e tráfego de rede para identificar atividades maliciosas ou intrusões. Os principais desafios em detecção de anomalias são o enorme volume e o formato não estruturado de dados, sem qualquer padronização.

Notavelmente, há algumas abordagens baseadas em *Deep Learning*, especialmente em Redes Neurais Recorrentes (RNNs) que já são utilizadas para detecção de anomalias em logs,

no entanto, dependentes da análise de logs para pré-processá-los. Ademais, ainda existem algumas limitações do uso de RNN para modelagem de logs devido à dificuldade em aprender e armazenar informações por muito tempo. RNNs convencionais tem problema quando são utilizadas para treinar com sequências muito longas.

O *Log Parser* ou análise de log, tem o propósito de transformar as mensagens de log brutas em uma sequência de eventos estruturados. Uma mensagem de log bruta consiste em duas partes: uma constante e outra variável. A parte constante constitui o texto fixo e representa a categoria do evento correspondente que permanece o mesmo para cada ocorrência do evento. Já a parte variável corresponde às informações providas pela ocorrência do evento do sistema. O objetivo da análise de log é separar automaticamente a parte constante e a parte variável de uma mensagem de log bruta para obter o significado dos logs.

As abordagens tradicionais dos analisadores de logs dependem de expressões regulares personalizadas para extrair os eventos de log específicos. No entanto, este método torna-se ineficiente e propenso a erros para sistemas modernos pelos seguintes motivos: 1) O volume de log cresce rapidamente e muda frequentemente; 2) Construir manualmente expressões regulares para um número tão grande de categorias de logs é complexo. Sendo assim, os métodos dos analisadores de logs existentes produzem um número de erros de análises que diminuem o desempenho na detecção de anomalias, resultando em exclusões de informações valiosas, o que pode levar a equívocos de semântica nas mensagens de logs.

Para lidar com as limitações existentes dos modelos baseados em RNN e com *Log Parser*, foi criado para fins deste trabalho o *RequestBERT-BiLSTM*. O modelo tem como finalidade ser uma possibilidade de proteção a servidores Web, o qual propõe uma estrutura supervisionada para detecção de ataques em requisições HTTP baseada em *Bidirectional Encoder Representations from Transformers (BERT)* sem a utilização de *Log Parser*.

Baseado no grande sucesso do *BERT* na modelagem de dados de texto sequencial (DEVLIN et al., 2018) o *BERT* foi incluído no projeto para capturar padrões de sequências de log utilizando sua estrutura. Sendo assim, espera-se que a incorporação contextual de cada requisição HTTP capture as informações essenciais para a detecção de ataques.

1.1 MOTIVAÇÃO

Tentar combater ataques a servidores Web ou ser capaz de detectar possíveis ataques é o que conduz o desenvolvimento desse projeto.

Atualmente, existem três abordagens populares para construir um Sistema de Detecção de Intrusão Web (WIDS). São abordagens baseadas em **assinatura**, **regras** e em **anomalias**. A detecção baseada em **assinatura** requer uma biblioteca que contenha os IDs dos ataques nos quais os logs das requisições são verificados, para então conferir se esses IDs aparecem em alguma requisição. Em caso afirmativo, a requisição é considerada maliciosa. A detecção baseada em **regras** é como a detecção baseada em assinaturas, mas não mantém uma biblioteca de IDs maliciosos. Em vez disso, essa abordagem verifica a requisição HTTP baseado em algumas regras já definidas, se uma requisição HTTP é benigna segue e se a requisição quebra uma ou várias regras, é considerada maliciosa. A detecção baseada em **anomalias** envolve o treinamento de um modelo para caracterizar as requisições e filtrá-las, possibilitando a detecção de novos tipos de ataques, ao contrário da detecção baseada em assinatura (ODUMUYIWA; CHIBUEZE, 2020).

As requisições HTTP podem ser monitoradas, para então serem extraídas informações importantes. Cada requisição HTTP registra informações e o conteúdo das mensagens trafegada do cliente para o servidor. Os campos de cabeçalho HTTP são uma lista de strings enviadas e recebidas pelo programa cliente e pelo servidor em cada requisição HTTP. Esses cabeçalhos geralmente são invisíveis para o usuário final, sendo processados ou registrados apenas pelo servidor e pelos aplicativos clientes. Eles definem como as informações são enviadas através da conexão (com Content-Encoding), a verificação e identificação da sessão do cliente (com os cookies do navegador e endereço IP), entre outros.

Baseado nisso, as requisições HTTP possuem bastantes elementos que podem influenciar qualquer uma das formas de um WIDS.

De acordo com grandes empresas que lidam diariamente com ataques e baseado nas informações colhidas por seus *Intrusion Prevention System (IPS)* que estão em execução em ambientes reais protegendo diversos tipos de clientes, parte significativa dos ataques tem como alvos servidores Web. A Figura 1 exibe a quantidade de ataques da segunda metade do ano de 2021 (FORTIGATE, 2022). Os servidores Web (*Apache, Nginx, IIs* etc.) em conjunto com linguagem PHP, estão entre aos principais alvos dos atacantes.

As técnicas de *deep learning* mais utilizadas na literatura, como nos estudos delineados por: (DU et al., 2017; MENG et al., 2019; ZHANG et al., 2019; GUO; YUAN; WU, 2021) apresentam maneiras para detecção de anomalias em logs com a utilização de *Log Parser* para realizar o tratamento inicial dos logs, no entanto, os trabalhos são restritos a base de dados específicas.

O *Log Parser* converte automaticamente cada mensagem de log em um modelo de log

específico, removendo parâmetros e mantendo as palavras-chave. Existem muitas técnicas de análise de log, incluindo mineração de padrões, agrupamento, modelagem de linguagem e heurísticas.

Trabalhos da literatura como: (GUO; YUAN; WU, 2021; DU et al., 2017; ZHANG et al., 2019) utilizam em seus modelos *Log Parser* e é possível notar que o processo descarta informações que podem ser indispensáveis para detecção de possíveis ataques. Já o RequestBERT-BiLSTM utiliza todo o texto para extrair a semântica dos logs sem a utilização de *Log Parser*.

Este trabalho inicia suas contribuições para com a comunidade científica observando que, partindo de uma linha de base apresentado em outros trabalhos, há possibilidade de melhorar o estado da arte do problema de detecção de ataques em requisições HTTP sem a utilização de *Log Parser*.

1.2 OBJETIVOS

Este trabalho investiga métodos de detecção de ataques em requisições HTTP visando proteger servidores Web. Posto isto, procurou-se adaptar modelos baseados em *Transformers* e sem a necessidade de realizar a análise de log nas requisições destinadas aos servidores Web.

1.3 OBJETIVOS ESPECÍFICOS

- Propor melhorias para detecção e prevenção do **estado da arte** de ataques a servidores Web, melhorando a metodologia adotada para proteção e permitindo uma melhor compreensão das requisições HTTP;
- Verificar a possibilidade de utilizar modelos baseados em técnicas de *deep learning* para detecção de ataques em requisições HTTP;
- Analisar a eficiência de ferramentas na análise de logs; e
- Com o uso de *Transformer*, desenvolver o primeiro modelo de detecção de ataques a servidores Web baseado em requisições HTTP, diferente dos modelos do estado da arte e realizar experimentos para compará-los.

1.4 APRESENTAÇÃO

Este trabalho está dividido em oito partes, o Capítulo 1 apresenta o problema, as motivações que levaram a esta pesquisa. No Capítulo 2 é feito uma explicação sobre detecção de ataques e demonstra as principais vulnerabilidades para servidores Web baseado na *OWASP top 10*. No Capítulo 3 discute alguns conceitos básicos fundamentais para o entendimento deste trabalho. O Capítulo 4 é feito um levantamento do estado atual do estudo e do problema na comunidade científica, sendo apresentados alguns projetos de destaque relacionados ao tema. No Capítulo 5 é apresentada a proposta deste trabalho, seguido pelos experimentos e resultados nos Capítulos 6 e 7, respectivamente. O Capítulo 8 apresenta as conclusões e contribuições desta pesquisa e discute algumas sugestões de trabalhos futuros.

2 DETECÇÃO DE ATAQUES

Servidores Web e aplicativos Web são alvos comuns de ataques. Os servidores Web geralmente são acessíveis por meio de *Firewalls* corporativos, e os aplicativos Web são geralmente desenvolvidos e operados sem seguir uma metodologia de segurança sólida. Ataques que exploram servidores Web ou extensões de servidor representam uma parte substancial do número total de vulnerabilidades.

Os Sistemas de Detecção de Intrusão, do inglês, *Intrusion Detection System - IDS* é um sistema que monitora uma rede em busca de eventos que violem as regras de segurança dessa rede. Infelizmente, é difícil manter os conjuntos de assinaturas de detecção de intrusão atualizados com relação ao grande número de vulnerabilidades criadas diariamente. Além disso, as vulnerabilidades podem ser introduzidas por aplicativos Web desenvolvidos internamente sem seguir nenhuma padronização e recomendações de segurança.

As vulnerabilidades dos servidores Web podem existir em aplicativos do lado do servidor e do lado do cliente. Os métodos de ataque variam conforme os resultados desejados pelo atacante. Uma vez que um aplicativo é vulnerável a um método específico de ataque, fornece ao invasor uma variedade de resultados de exploração. As explorações desejadas podem incluir *code execution, DoS, information disclosure, defacement, session hijacking, loss of trust e data manipulation etc.*.

A detecção de ameaças visa identificar as ameaças potenciais e os vetores de ataque de um sistema — essas informações permitem que as equipes analisem e determinem as medidas para atenuar os riscos. Uma estrutura de modelagem de ameaças pode estruturar esse processo e melhorar a capacidade de uma organização de identificar ameaças. Os modeladores de ameaças adotam a perspectiva de um hacker para avaliar os danos que podem causar. Eles analisam minuciosamente a arquitetura de software e o contexto de negócios para obter *insights* aprofundados sobre o sistema.

Neste cenário existem duas estruturas e metodologias de modelagem de ameaças: *MITRE ATT&CK Framework* e *OWASP top 10*.

2.1 MITRE ATT&CK

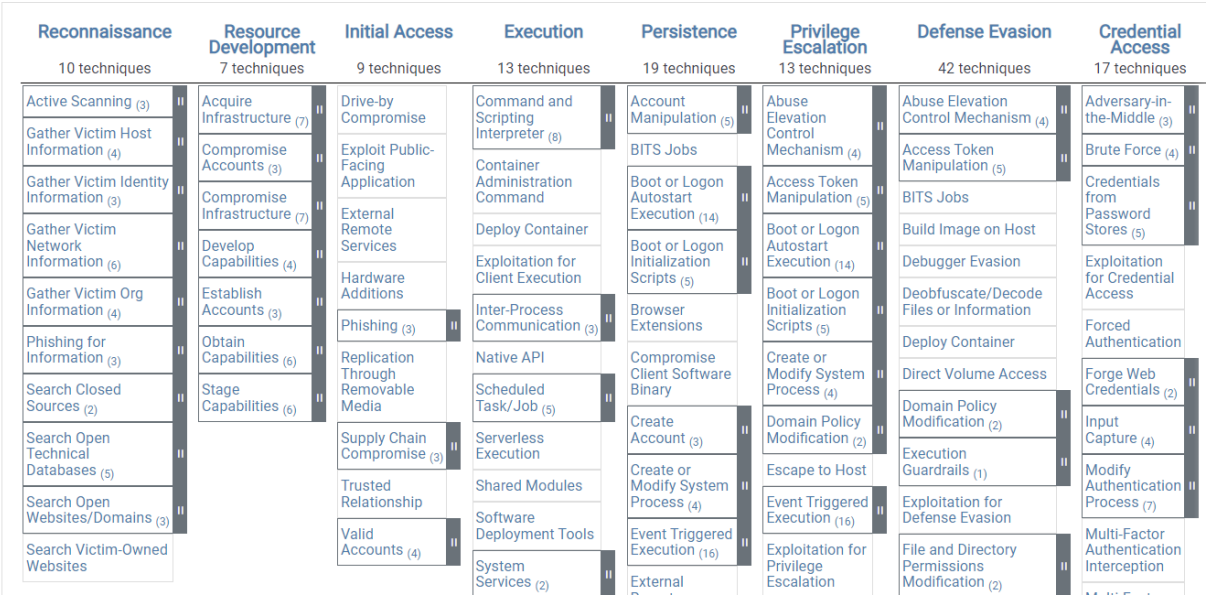
A *MITRE* introduziu o *ATT&CK ((Adversarial Tactics, Techniques Common Knowledge* – Táticas, Técnicas e Conhecimento Comum de Adversários) em 2013 como uma forma de descrever e categorizar os comportamentos dos invasores. O *ATT&CK* é uma lista estruturada de comportamentos conhecidos de invasores, compilados em táticas e técnicas, expressos em várias matrizes. Como essa lista é uma representação abrangente dos comportamentos que os invasores utilizam quando comprometem as redes, ela é útil para diversas medidas ofensivas e defensivas.

As matrizes do *MITRE ATT&CK* contém várias táticas e técnicas associadas ao seu tema.

A matriz *Enterprise* é formada por técnicas e táticas que se aplicam aos sistemas *Windows, Linux e/ou MacOS* e ao que os invasores fazem antes de tentarem explorar uma determinada rede ou sistema alvo. A matriz *Mobile* contém táticas e técnicas aplicadas em dispositivos móveis e a *ICS* contém táticas e técnicas relacionadas a sistemas de controle industrial.

Ao olhar para o *ATT&CK* na forma de uma matriz, os títulos das colunas na parte superior são as táticas que são, basicamente, as categorias das técnicas. Táticas são o que os invasores estão tentando alcançar, enquanto as técnicas individuais são como eles realizam essas etapas ou metas.

Figura 2 – Matriz *Enterprise* com algumas táticas e técnicas.



Fonte: (MITRE, 2022)

Por exemplo, uma das táticas é a escalação de privilégios. Para que um invasor consiga executar escalação de privilégios com sucesso, ele utilizará uma ou mais das técnicas listadas na

coluna escalação de privilégios. Além disso, o *ATT&CK* fornece diversos detalhes sobre cada técnica, incluindo descrições, exemplos, referências e sugestões para mitigação e detecção.

Portanto, essas informações podem ajudar a garantir que uma ameaça potencial seja devidamente identificada durante a modelagem de ameaças e fornecem orientações sobre como mitigar o risco potencial usando uma combinação de estratégias de detecção e mitigação.

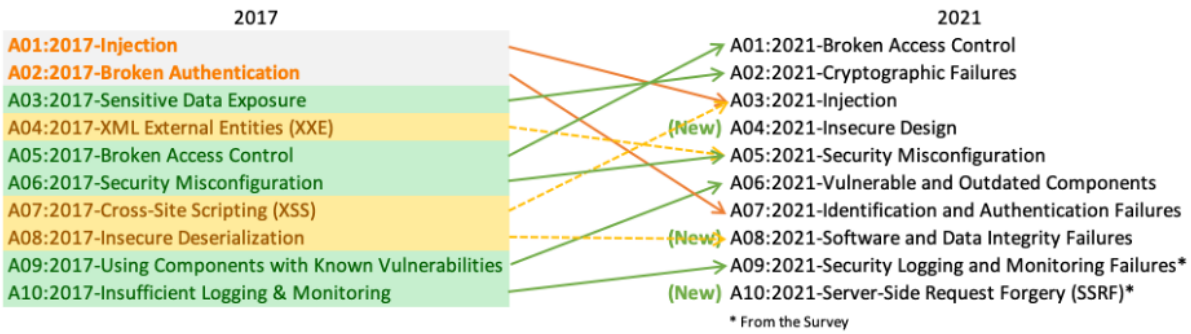
2.2 OWASP TOP 10 VULNERABILIDADES

Segundo *OWASP*, uma comunidade online, onde apresenta pesquisas para auxiliar na melhoria e testes para o funcionamento da segurança arquitetural dos projetos de sistemas Web. Em 2021, realizou um levantamento global com os 10 principais tipos de ataque estudados pela instituição para aplicações Web, e demonstrou as maiores vulnerabilidades apresentadas nas aplicações conforme a Figura 3. E compreender as vulnerabilidades comuns em aplicativos Web pode ajudar as empresas a se prepararem melhor para proteger seus dados contra os ataques (OWASP, 2020). O objetivo do *OWASP* top 10 não é ser apenas uma lista e sim avaliar cada classe de falha usando a metodologia *OWASP Risk Rating* e fornecer diretrizes, exemplos, melhores práticas para prevenir ataques e referências para cada risco.

Ao aprender as falhas no gráfico *OWASP* Top 10 e como resolvê-las, os desenvolvedores de aplicativos podem tomar medidas concretas em direção a um aplicativo mais seguro.

Este capítulo terá como objetivo fornecer aos leitores uma melhor compreensão sobre as 5 principais vulnerabilidades segundo a lista *OWASP*.

Figura 3 – *OWASP* top 10 é uma lista dos 10 riscos de segurança de aplicativos web mais comuns pesquisados pelo OWASP



Fonte: (OWASP, 2020)

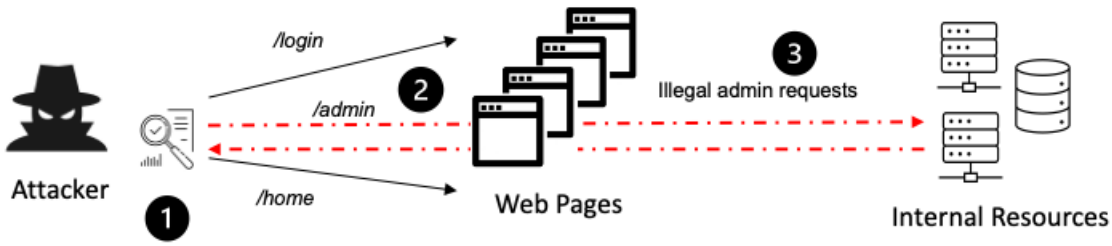
2.2.1 *Broken Access Control*

Quebra de controle de acesso é um termo usado para várias vulnerabilidades que permitem aos invasores explorar e representar usuários de aplicativos Web. Existem vários métodos nos quais os invasores podem obter credenciais de usuário ou sequestrar sessões de usuário para poder representar esses usuários, como credenciais fracas ou previsíveis, etc. (BACH-NUTMAN, 2020).

No cenário a seguir, o invasor usa técnicas de navegação forçada para explorar um diretório estático desprotegido no sistema Web.

1. O invasor usa uma ferramenta de verificação automatizada para procurar recursos não vinculados no sistema Web e encontra o seguinte recurso desprotegido: **/admin**
2. O invasor inicia um ataque de navegação forçada no sistema Web para verificar se são necessários direitos administrativos para acessar a página: **https://example.com/admin**.
3. O invasor acessa a página de administração como um usuário não autenticado e executa ações não autorizadas.

Figura 4 – Cenário de ataque de quebra de controle de acesso



Fonte: (F5, 2022)

Mitigação contra este tipo de ataque incluem *hashing* de senhas, garantir que os IDs de sessão não sejam expostos nas URLs, configurações de tempo limite nas sessões, obrigatoriedade de sessões serem recriadas após um certo período, evitar que as senhas não sejam enviadas em conexões sem criptografia, etc.

2.2.2 *Cryptographic Failures*

Uma falha de criptografia é uma vulnerabilidade crítica de segurança de aplicativos Web que expõe dados confidenciais do aplicativo devido a um algoritmo criptográfico fraco ou inexistente.

Os invasores visam geralmente dados confidenciais, como senhas, números de cartão de crédito e informações pessoais. A falha criptográfica é a causa principal da exposição de dados confidenciais. Segundo o *OWASP* 2021, proteger seus dados contra falhas criptográficas tornou-se mais importante do que nunca.

No cenário de ataque a seguir, um invasor usa uma tabela *Rainbow* para quebrar *hashes* de senhas.

1. O invasor obtém acesso à rede de uma organização;
2. O invasor usa uma falha de aplicativo para recuperar um banco de dados de senhas;
3. Como o banco de dados utilizou *hashes* sem saltos para criptografar senhas, o invasor pode usar uma tabela *Rainbow* para expor as senhas.
4. O invasor usa ferramentas de preenchimento de credenciais para testar pares de credenciais em outros sites.

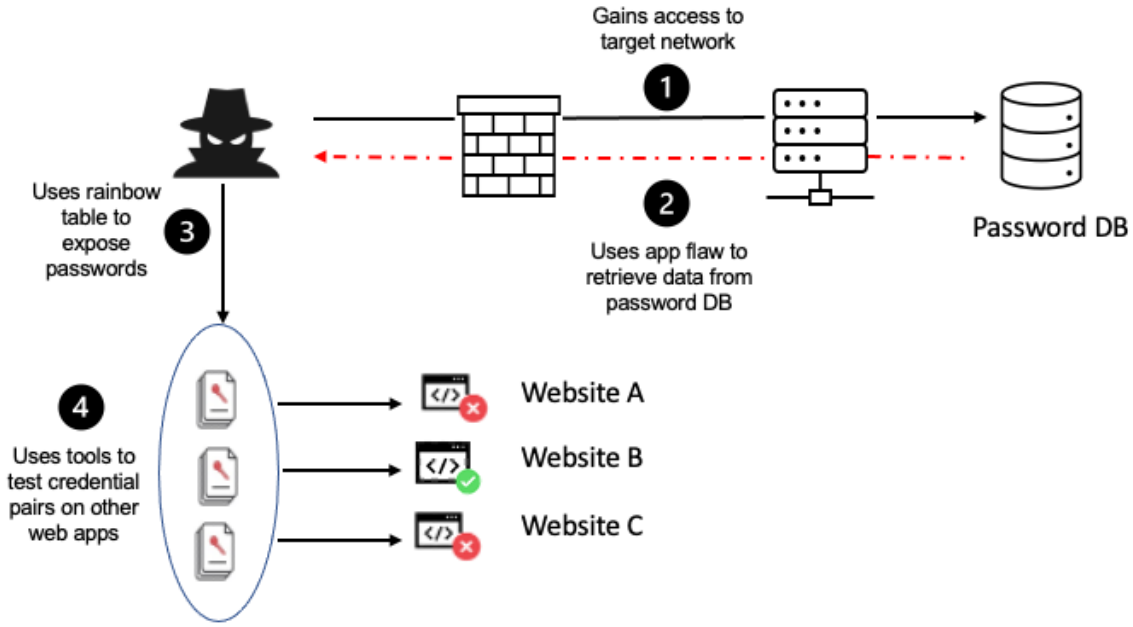
Para evitar este tipo de ataque, no mínimo, é recomendado realizar algumas ações: criptografar todos os dados confidenciais salvos fisicamente, garantir que algoritmos, protocolos e chaves padrões estejam atualizados e fortes, utilizar gerenciamento de chaves adequado, criptografar todos os dados em trânsito com protocolos seguros como: *Transport Layer Security (TLS)*.

2.2.3 *Injection Attack*

Os ataques de injeção são um dos ataques mais perigosos em que um invasor simplesmente envia dados maliciosos para fazer com que o aplicativo os processe e faça algo que não deveria. Vulnerabilidades de injeção são predominantes, especialmente em código legado que não valida ou limpa a entrada fornecida pelo usuário.

As tecnologias de aplicativos comuns que podem ser vítimas de um ataque de injeção são as seguintes: *SQL, LDAP, XPathName*, etc. Os invasores geralmente exploram falhas de injeção

Figura 5 – Cenário de ataque de falha de criptografia



Fonte: (F5, 2022)

injetando um comando do sistema operacional, consulta SQL ou até mesmo um script completo em um parâmetro, cabeçalho, URL ou outra forma de dados que um aplicativo recebe.

A injeção de SQL é um método muito comum de ataque de injeção. Para executar a injeção de SQL, um invasor modifica parte de uma string de consulta enviada a um banco de dados SQL para executar uma ação maliciosa, como listar todos os nomes de usuário e senhas.

No cenário de ataque a seguir, o invasor modifica um valor de parâmetro em seu navegador para enviar uma instrução SQL criada para o aplicativo vulnerável:

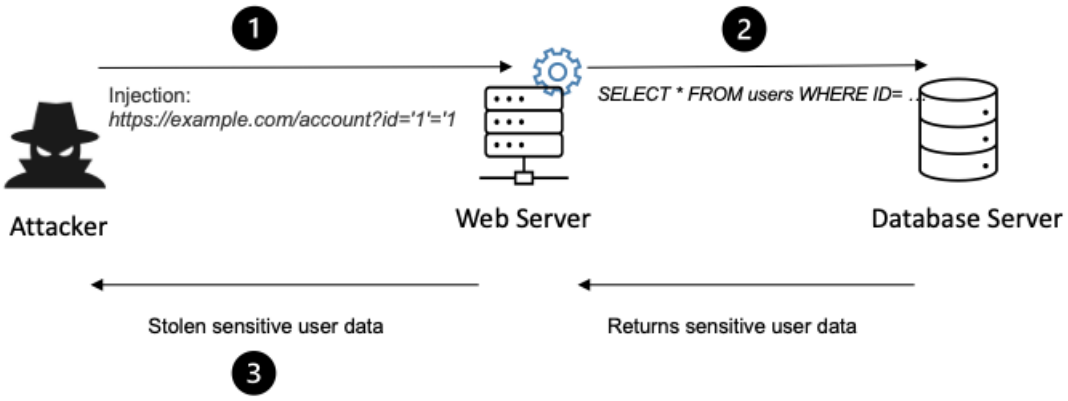
1. Um invasor injeta um comando em um elemento vulnerável do aplicativo. Por exemplo, **https://example.com/account?id='1'='1**

2. O servidor de aplicativos Web executa a injeção.

3. O invasor obtém acesso não autorizado e pode causar uma negação de serviço ou roubar dados confidenciais. No caso do exemplo anterior, o invasor pode obter todos os campos de dados de todos os usuários, em vez de um usuário específico.

A prevenção deste tipo de ataque pode ser realizada seguindo algumas recomendações: manter os dados separados dos comandos e consultas, validação de entrada de dados do lado do servidor e atenção redobrada no tratamento dos caracteres especiais nas áreas de texto.

Figura 6 – Cenário de ataque de SQL Injection



Fonte: (F5, 2022)

2.2.4 *Insecure Design*

O design inseguro é focado nos riscos associados a falhas no design e na arquitetura dos sistemas. Ele se concentra na necessidade de modelagem de ameaças, padrões de design seguros e princípios.

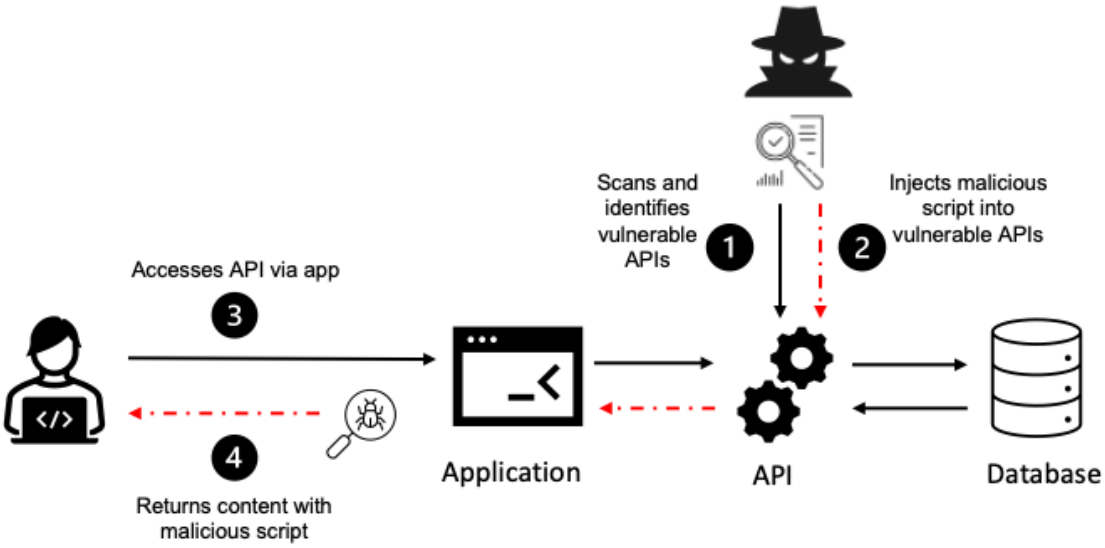
Um design inseguro não pode ser corrigido por uma implementação perfeita, pois, os controles de segurança necessários nunca foram criados para se defender contra os ataques específicos. Para explorar o design inseguro, os invasores podem ameaçar os fluxos de trabalho do modelo no *software* para revelar uma ampla gama de vulnerabilidades e fraquezas.

No cenário de ataque a seguir, o invasor explora uma API mal projetada que não filtra adequadamente a entrada.

1. O invasor procura APIs vulneráveis e identifica uma API que não filtra adequadamente a entrada;
2. O invasor injeta um script malicioso na API vulnerável;
3. O navegador da vítima acessa a API por meio do aplicativo; e
4. O navegador carrega o conteúdo com o script malicioso.

A prevenção deste ataque seguem alguns passos: estabelecer e utilizar bibliotecas baseado em padrões de projeto seguros, integrar verificações em cada camada do aplicativo (*front-end e back-end*), desenvolver testes de unidade e integração para validar se todos os fluxos críticos são resistente e limitar o consumo de recursos por usuário ou serviço.

Figura 7 – Cenário de ataque de design inseguro.



Fonte: (F5, 2022)

2.2.5 *Security misconfiguration*

As vulnerabilidades de configuração incorreta de segurança ocorrem quando um componente de aplicativo Web é suscetível a ataques devido a uma configuração incorreta ou opção de configuração insegura.

Vulnerabilidades de configuração incorreta são pontos fracos de configuração que podem existir em componentes e subsistemas de *software* ou na administração do usuário. Por exemplo, o *software* de servidor Web pode ser fornecido com contas de usuário padrão que um invasor pode usar para acessar o sistema ou o software pode conter arquivos de amostra, como arquivos de configuração e scripts que um invasor pode explorar. Além disso, o software pode ter serviços desnecessários ativados, como a funcionalidade de administração remota.

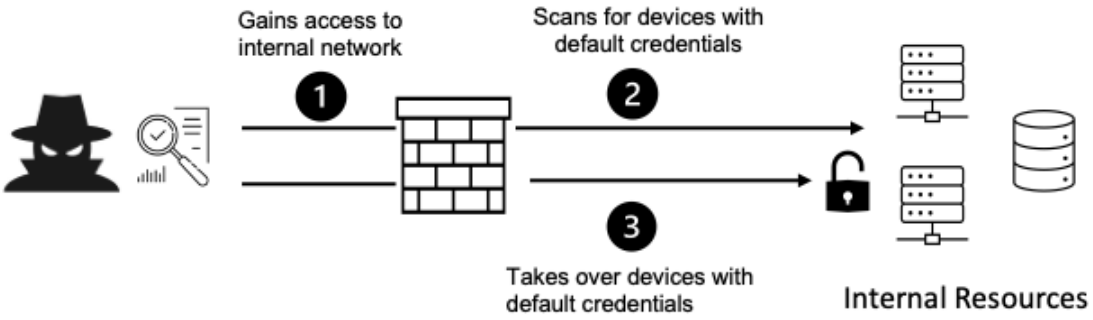
Por exemplo, os seguintes tipos de ataque podem ter como alvo vulnerabilidades de configuração incorreta: *Brute force/credential stuffing, Code injection, Buffer overflow, Command injection, Cross-site scripting (XSS) e Forceful browsing*

No cenário de ataque a seguir, o invasor explora dispositivos de rede que usam credenciais padrão.

1. O invasor obtém acesso à rede interna de uma organização;
2. O invasor verifica a rede em busca de dispositivos e realiza verificações de dicionário para determinar quais dispositivos estão usando credenciais padrão; e

3. O invasor faz login nos dispositivos vulneráveis usando as senhas padrão e assume o controle.

Figura 8 – Cenário de ataque de configuração incorreta de segurança.



Fonte: (F5, 2022)

Algumas maneiras de tentar impedir este ataque: implementar processos de instalação de aplicativos seguros, controle de qualidade e produção devem ser configurados de forma idêntica e remover ou não instalar recursos e estruturas não utilizados.

2.3 CONSIDERAÇÕES

O impacto nos negócios e usuários varia de acordo com o tipo de ataque e a sensibilidade dos dados, mas acaba deixando empresas em perdas financeiras e usuários com dados sensíveis expostos. Este capítulo mostrou as 5 principais vulnerabilidades *OWASP* que podem ser descobertas por meio das requisições HTTP. Também foi explicado o funcionamento do *MITRE ATT&CK* que é uma lista exaustiva de todas as técnicas de ataque em potencial, mas abrange uma gama impressionante de ameaças e oferece critérios claros para identificar vulnerabilidades. Portanto, monitorando e conseguindo detectar essas vulnerabilidades descritas podem ajudar empresas e usuários finais a se proteger desses ataques e manter seus ativos e dados preservados.

3 REVISÃO DA LITERATURA

Este capítulo tem o propósito de apresentar a revisão bibliográfica dos conceitos e abordagens relacionados ao desenvolvimento desta pesquisa, descrevendo suas características e formas de aplicação. Primeiramente, serão abordados os conceitos de requisições HTTP que norteiam toda a ideia deste projeto, posteriormente, detecção de anomalias, *Log Parser* e os métodos que serão utilizados neste trabalho, conceitos de aprendizagem profunda, explicação de Processamento de Linguagem Natural e a como acontece a representação textual e, por fim, *Transformers* e *BERT*.

3.1 REQUISIÇÕES HTTP

HTTP é um protocolo baseado em *streams* de texto. Em suma, o cliente abre um *socket* para falar com o servidor e, nesse *socket*, envia requisições *socket(requests)*, as quais o servidor responderá com respostas *socket(responses)*.

Uma requisição HTTP é feita por um cliente, para um host nomeado, localizado em um servidor. O objetivo da requisição é acessar um recurso no servidor.

Para efetuar a requisição, o cliente utiliza componentes de uma *URL (Uniform Resource Locator)*, que inclui as informações necessárias para acessar o recurso.

Uma requisição HTTP composta corretamente contém os seguintes elementos:

- Linha de Requisição;
- Cabeçalhos HTTP;
- Corpo da mensagem.

3.1.1 Linha de Requisição

A linha de requisição é a primeira linha na mensagem da requisição. É composta por pelo menos três itens:

- Método HTTP: Um verbo (como *GET, PUT ou POST*) ou um nome (como *HEAD ou OPTIONS*), que descrevem a ação a ser executada;
- Path-URL: Normalmente uma URL, ou o caminho absoluto do protocolo;

- Versão HTTP: Define a estrutura do restante da mensagem, atuando como um indicador da versão esperada para uso na resposta.

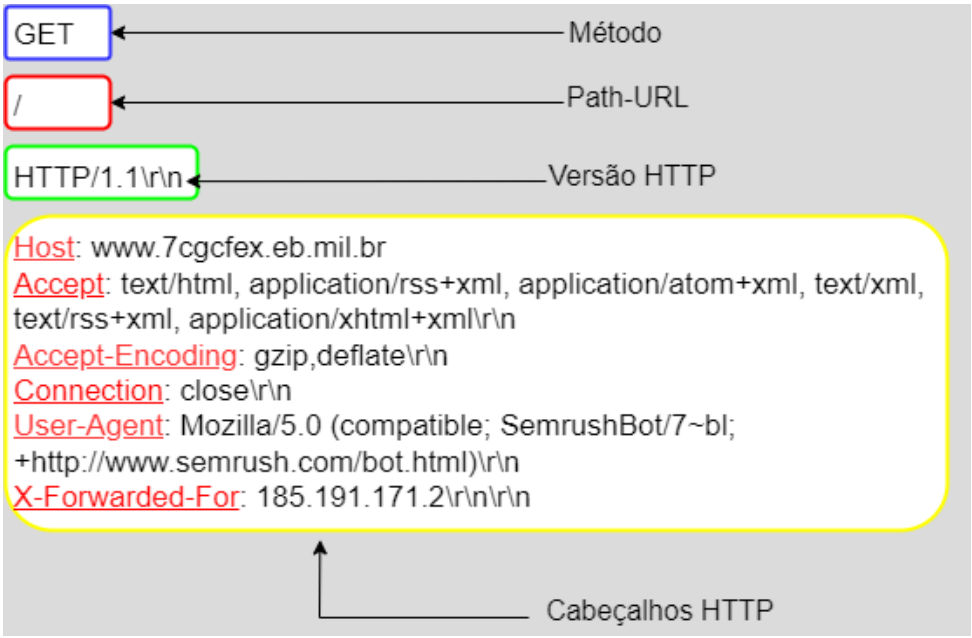
O formato da linha de requisição segue conforme a Figura 9, onde SP = espaço em branco.

Figura 9 – Exemplo da composição de uma Linha de Requisição contida na requisição HTTP.

Fonte: Elaborado pelo autor, 2022.

O texto da Figura 9 é lido linha a linha, sendo cada linha separada por CRLF, ou seja, no final de cada linha é necessário uma quebra-de-linha com: \r\n.

Figura 10 – Componentes da requisição HTTP na sequência: linha de requisição nas caixas de texto de cores azul, vermelha e verde, já os cabeçalhos HTTP estão na caixa de texto amarelo. O próximo componente seria o body, porém como não há dados nesta requisição, não houve nenhuma informação.



Fonte: Elaborado pelo autor, 2022.

Na Figura 10 os campos do cabeçalho estão identificados em vermelho e sublinhados. Toda a requisição HTTP é utilizada para extração de informações pelo nosso modelo, então para as Figuras 9 e 10 todo o conteúdo é aplicado.

3.1.2 Cabeçalhos HTTP

Os possíveis cabeçalhos são muitos. Alguns exemplos são: *Accept-Charset, Accept-Encoding e User-Agent*. Os cabeçalhos HTTP são escritos em uma mensagem para fornecer ao destinatário

informações sobre a mensagem, sobre o remetente e a maneira como o remetente deseja se comunicar com o destinatário. Cada cabeçalho HTTP é composto por um nome e um valor. As especificações do protocolo HTTP definem o conjunto padrão de cabeçalhos HTTP e descrevem como usá-los corretamente (MDN, 2021). Os campos de cabeçalho HTTP são uma lista de strings enviadas e recebidas pelo programa cliente e pelo servidor em cada requisição HTTP. Esses cabeçalhos geralmente são invisíveis para o usuário final, sendo processados ou registrados apenas pelo servidor e pelos aplicativos clientes. Eles possuem algumas características: definir como as informações enviadas através da conexão são codificadas (como em *Content-Encoding*), a verificação e identificação da sessão do cliente (como nos *cookies* do navegador) e endereço IP. Alguns tipos de cabeçalhos podem ser observado nas Figuras 10 e 11.

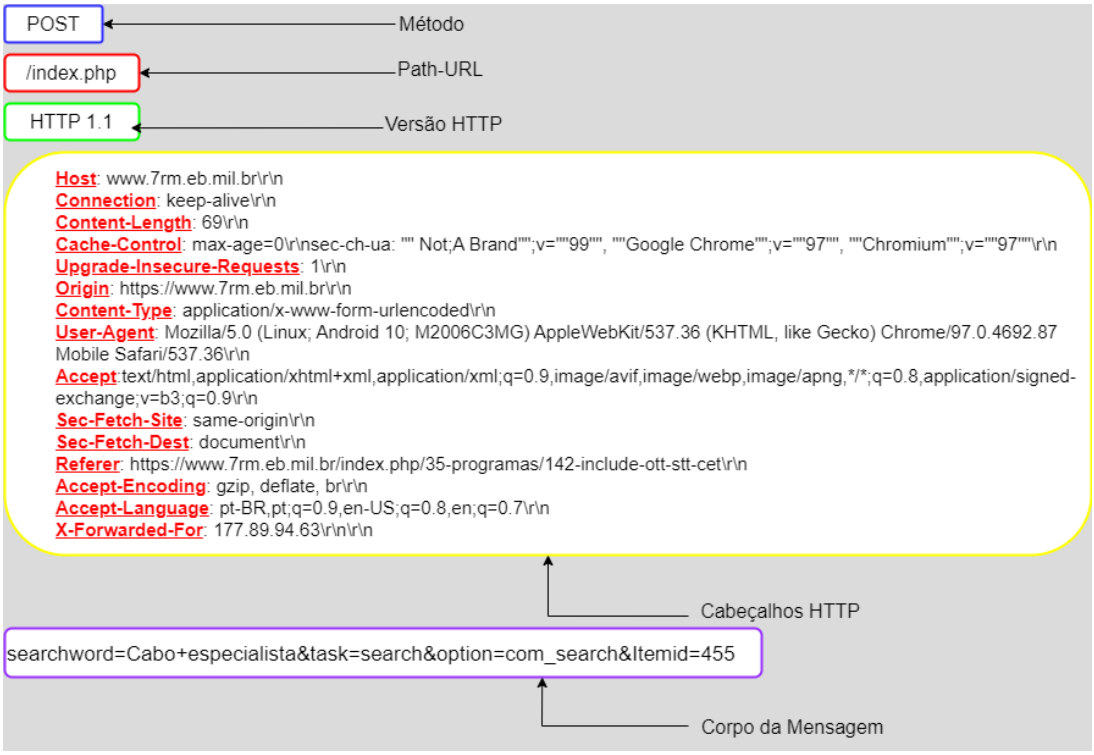
3.1.3 Corpo de mensagem

Os corpos das mensagens são apropriados para alguns métodos de solicitação e inadequados para outros. Por exemplo, uma solicitação com o método *POST*, envia dados de entrada para o servidor conforme a Figura 11 em que possui um corpo de mensagem contendo os dados. Uma solicitação com o método *GET*, solicita ao servidor o envio de um recurso, logo, não possui um corpo de mensagem como já mostrado na Figura 10 (MDN, 2021).

Além disso, o corpo da requisição pode ser dividido, a grosso modo, em duas categorias:

- Corpo de recurso-simples: consistindo em um único arquivo, definido pelos dois cabeçalhos: *Content-Type e Content-Length*; e
- Corpo de recurso-múltiplo: consistindo em um corpo de múltiplas partes, cada uma contendo uma porção diferente de informação. Este é tipicamente associado a formulários HTML.

Figura 11 – Componentes da Requisição HTTP com o Body.



Fonte: Elaborado pelo autor, 2022.

3.2 DETECÇÃO DE ANOMALIAS

Com o aumento significativo de dispositivos computacionais nos últimos anos, a quantidade de dados transmitidos e armazenados cresceu de forma alarmante. Diante disso, os logs de sistemas são artefatos essenciais para a execução das técnicas de detecção de anomalias. Eles registram os estados e eventos significativos do sistema ajudando a depurar os comportamentos que não são esperados.

A detecção de anomalias refere-se ao problema de encontrar padrões em dados que não correspondam ao comportamento esperado. Esses padrões com inconformidades são frequentemente chamados de: anomalias, *outliers*, observações discordantes, exceções, aberrações, surpresas, peculiaridades ou contaminantes em diferentes domínios de aplicação. Destes, as anomalias e *outliers* são dois termos usados mais comumente no contexto de detecção de anomalias; às vezes de forma intercambiável. Além disso, encontra uso extensivo em uma ampla variedade de aplicativos como detecção de fraudes em cartões de crédito, detecção de intrusão em segurança cibernética, detecção de falhas em sistemas críticos de segurança e vigilância militar (CHANDOLA; BANERJEE; KUMAR, 2009).

Ao contrário dos problemas e na maioria das tarefas que seguem alguns padrões regulares

ou evidentes, a detecção de anomalia aborda eventos minoritários, imprevisíveis/incertos e raros, trazendo complexidades únicas para os métodos de detecção. As anomalias estão associadas a muitas incógnitas, por exemplo, instâncias com comportamentos abruptos desconhecidos, estruturas de dados e distribuições. Elas permanecem desconhecidas até que realmente ocorram, como novos ataques terroristas, fraudes e invasões de rede. As anomalias são comportamentos irregulares, portanto, uma classe de anomalia pode apresentar características anormais completamente diferentes de outra classe de anomalia. Elas são tipicamente instâncias de dados raras, contrastando com instâncias normais que respondem muitas vezes por uma proporção esmagadora dos dados (PANG et al., 2022).

Existem três tipos completamente diferentes de anomalia e foram exploradas por (CHANDOLA; BANERJEE; KUMAR, 2009):

- **Anomalias Pontuais**: são ocorrências individuais que geralmente representam uma irregularidade que pode não ter uma interpretação particular;
- **Anomalias Condicionais**: também conhecidas como **anomalias contextuais**, também se referem a instâncias anômalas individuais, mas em um contexto específico, ou seja, instâncias de dados são anômalas no contexto específico, caso contrário normais. Os contextos podem ser muito diferentes em aplicações reais, por exemplo, temperatura repentina, queda/aumento em um determinado contexto temporal, ou transações rápidas na madrugada com cartão de crédito; e
- **Anomalias de Grupo**: também conhecidas como **anomalias coletivas**, são um subconjunto de instâncias de dados anômalos. Individualmente, aparecem como instâncias normais, e quando observadas em grupo, exibem características incomuns. Ex: saques em locais distantes num período curto.

3.3 LOG PREPROCESSING

O pré-processamento de log é etapa inicial que lida com o tratamento e preparação dos dados para serem utilizados adequadamente no modelo.

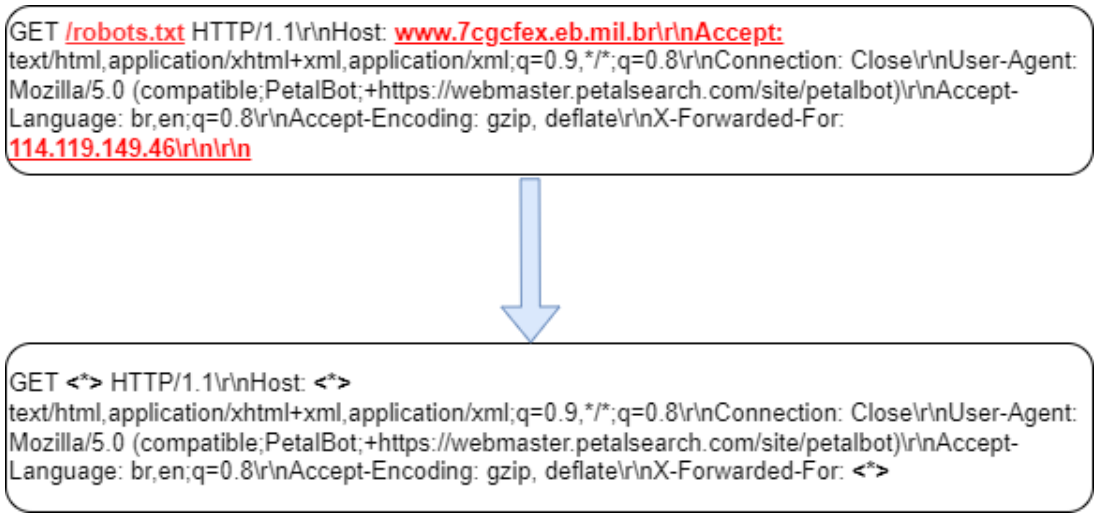
3.3.1 *Log Parsing* ou Análise de Log

O *Log Parsing* ou análise de log é a etapa inicial do pré-processamento de log e seu objetivo é deixar os logs não estruturados mais compreensíveis para serem utilizados como entrada em outro processo. Cada mensagem de log bruto pode ser dividida em duas partes: **constante e variável** (HE et al., 2018). A parte constante é um modelo com textos fixos que permanecem os mesmos em cada entrada de log, enquanto a parte variável contém parâmetros que mudam dinamicamente a cada ocorrência do evento. A parte constante pode ser referida alternadamente como chave de log (DU et al., 2017), modelo de log (MENG et al., 2019) ou evento de log (ZHANG et al., 2019). Sendo assim, o objetivo da análise de log é extrair a parte constante e a parte variável das mensagens de log (GUO; YUAN; WU, 2021).

No entanto, nos casos das requisições HTTP que variam de tamanho é complexo para qualquer analisador de log obter a parte constante e a parte variável.

A Figura 12 mostra um exemplo de uma requisição e o seu resultado após a utilização do *Log Parser DRAIN* (HE et al., 2017). É possível notar que somente alguns campos foram compreendidos como partes variáveis, além disso, houve equívocos por parte do analisador que considerou os campos **Accept-Language, Connection e User-Agent** como elementos variáveis de **Host** e considerou também o método *GET* como parte fixa.

Figura 12 – Exemplo de uma requisição HTTP do dataset Log-Security e o resultado após a utilização do *Log Parser* (*DRAIN*). As chaves de log extraídas pelo *DRAIN* são as mensagens em vermelho que indicam: path da url, a url acessada e o ip de origem.



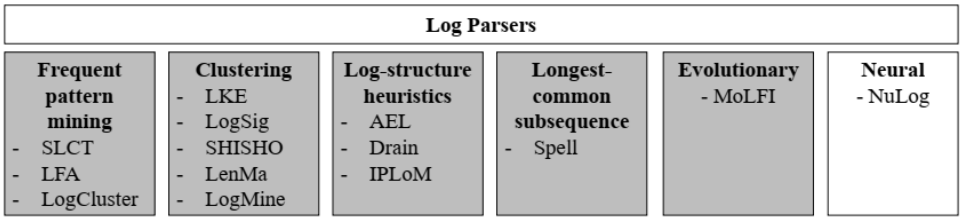
Fonte: Elaborado pelo autor, 2022.

No trabalho, (NEDELKOSKI et al., 2021) os autores resumiram 13 analisadores baseados em suas características existentes e categorizou em seis grupos: **clustering, mineração de**

padrões frequentes, evolutiva, heurística e subsequência comum mais longa. O grupo Neural foi o que ele implementou e sugeriu como uma nova abordagem em seu artigo.

Nesta dissertação serão escolhidos os seguintes (*Log Parser*: (*Drain, AEL, LogSig e LFA*) cada um com características diferentes conforme a Figura 13. Todos os analisadores de log são amplamente utilizados, no entanto, os escolhidos neste trabalho podem ter sua eficiência comprovada em estudos existentes com conjuntos de dados reais (ZHU et al., 2018; HE et al., 2016; HE et al., 2016).

Figura 13 – Categorias de Analisadores de log de acordo com suas características. Os escolhidos para este trabalho foram: (*Drain, AEL, LogSig e LFA*). O método *NuLog* foi o método desenvolvido no trabalho (NEDELKOSKI et al., 2021).



Fonte: (NEDELKOSKI et al., 2021)

Os métodos de *Log Parser* podem ser divididos baseados nas seguintes características:

- ***Clustering***: ou seja, cluster. O propósito nesse método é que as mesmas categorias de mensagens coincidam em grupos semelhantes. O *LogSig* (TANG; LI; PERNG, 2011) é um algoritmo baseado em assinatura de mensagem a partir de mensagens textuais de log. Ao pesquisar as assinaturas de mensagens mais representativas, o *LogSig* categoriza-as em um conjunto de grupos de eventos, com isso agrupando-as em cluster (NEDELKOSKI et al., 2021).

- ***Frequent pattern mining***: ou seja, mineração de padrões frequentes. Esse método assume que um tipo de mensagem é um padrão frequente de tokens que aparecem ao longo dos logs. Um padrão frequente é um conjunto de itens que ocorre com frequência em um conjunto de dados. Os procedimentos envolvem: a criação de conjuntos frequentes, agrupamento das mensagens de log e extração de tipos de mensagem. Portanto, a mineração de padrões frequentes é uma abordagem direta para análise automatizada de logs (ZHU et al., 2019).

  O *LFA* (NAGAPPAN; VOUK, 2010) segue um procedimento de análise: 1) Percorre os dados de log por várias passagens; 2) Constrói conjuntos de itens frequentes (por exemplo, tokens) em cada passagem; 3) Agrupa mensagens de log em vários clusters e;

4) Extrai modelos de eventos de cada cluster. Além disso, o *LFA* considera a distribuição de frequência do token em cada log ao invés de todos os dados de log para analisar mensagens raras.

- ***Evolutionary***: utiliza uma abordagem evolucionária para encontrar o conjunto ótimo de Pareto, ou seja, é um estado em que os modelos das mensagens estão alocados da forma mais eficiente possível (NEDELKOSKI et al., 2021);
- ***Heuristics***: eles geralmente exploram diferentes propriedades da estrutura dos logs, com isso produzindo assim os melhores resultados entre as diferentes técnicas adotadas (NEDELKOSKI et al., 2021). O *AEL* (JIANG et al., 2008a) separa mensagens em vários grupos, comparando as ocorrências entre tokens constantes e tokens variáveis. Já o *DRAIN* (HE et al., 2017) assume que no início dos logs as palavras não variam muito, então ele usa essa suposição para criar uma árvore de profundidade fixa que pode ser facilmente modificada para novos grupos;
- ***Longest-common sub-sequence***: ou seja, subsequência comum mais longa. O *SPELL* (DU; LI, 2016) usa o algoritmo de subsequência comum mais longo para extrair dinamicamente modelos e parâmetros de mensagens estruturadas em um modo de streaming em conjunto de dados não estruturados.

3.3.1.1 Drain

É um método de análise de log baseado em árvore. Quando log novo chega, o *DRAIN* realiza o pré-processamento baseado em expressões regulares configuradas segundo o conhecimento do domínio. Em seguida, um grupo de log é pesquisado seguindo as regras codificadas nos nós internos da árvore. Se um grupo de log adequado for encontrado, a mensagem de log será correspondida com o evento de log armazenado nesse grupo de logs. Caso contrário, um novo grupo de log será criado com base na mensagem de log (HE et al., 2017).

3.3.1.2 AEL

AEL (JIANG et al., 2008b) compreende quatro etapas: **anonimizar, tokenizar, categorizar e reorganizar**. Abstrair linhas de log para eventos é uma tarefa desafiadora e demorada.

A etapa de **anonimização** utiliza heurística para reconhecer tokens dinâmicos nas linhas dos logs. Dado que os tokens dinâmicos são reconhecidos, eles são substituídos por um token genérico ($v). Heurísticas podem ser adicionadas ou removidas desta etapa com base no conhecimento do domínio.

A etapa de **tokenização** separa as linhas de log anonimizadas em diferentes grupos (ou seja, *bins*) conforme o número de palavras e parâmetros estimados em cada linha de log. O uso de vários *bins* limita a espaço de busca da etapa seguinte (ou seja, a etapa de categorização).

A etapa de **categorização** compara as linhas de log em cada *bin* e as abstrai para a execução correspondente dos eventos. Os eventos de execução inferidos são armazenados em um banco de dados de eventos de execução para referências futuras.

Como a etapa de **anonimização** usa heurística para identificar informações dinâmicas em uma linha de log, há uma chance que possa perder algumas informações dinâmicas. As informações perdidas resultarão na abstração de várias linhas de log para vários eventos de execução que são semelhantes. A etapa de reconciliação aborda essa situação. Todos os eventos de execução são reexaminados para identificar quais devem ser mescladas para cada grupo específico.

3.3.1.3 LogSig

O *LogSig* (TANG; LI; PERNG, 2011) foi criado com o propósito de analisar eventos de mensagens de log textuais. O algoritmo *LogSig* tenta encontrar um número de *k* assinaturas de mensagens que correspondam a todas as mensagens dadas, em que k é especificado pelo usuário.

O *LogSig* trabalha na geração de eventos de sistema a partir de dados brutos de log, onde seu objetivo é agrupar as mensagens de log. Um problema é que, se a maioria dos termos de uma mensagem de log são termos de parâmetro, então este tipo de mensagem de log pode não ter muitas palavras comuns correspondentes. O algoritmo *LogSig* foi desenvolvido para lidar com várias categorias de dados de log sem muito conhecimento prévio e sem o formato de registro.

3.3.1.4 LFA

No *LFA* (NAGAPPAN; VOUK, 2010) as frequências de token são comparadas em cada mensagem de log em vez de ser em todas as mensagens de log, portanto, os parâmetros podem ser identificados distinguindo as frequências de token em uma mensagem de log.

Na primeira passagem, é construído um resumo de dados das palavras em um o arquivo de registro. Esta tabela de frequência é que contém o número de vezes que uma determinada palavra ocorre em uma determinada posição na linha de registro. Sendo assim, as linhas na tabela são as palavras, e as colunas são as posições em cada linha de log. No final da primeira passagem, é concluída a construção da tabela. Na segunda passagem, é examinada cada linha de log novamente. Nesta etapa é realizada uma pesquisa na tabela para cada palavra na linha para então ser escolhida a frequência dessa palavra nessa posição no log. Isso é feito para todas as palavras de log. Em seguida, busca-se por palavras que ocorrem em um número semelhante de vezes. Visto que é encontrado o cluster com o maior número de palavras, é possível encontra a palavra com a menor frequência, então qualquer palavra na linha com uma frequência maior ou igual a este valor seria uma palavra constante na linha de log.

3.4 APRENDIZAGEM PROFUNDA

A aprendizagem profunda é uma abordagem de aprendizado de máquina que emprega algoritmos inspirados na estrutura e função do cérebro. Nos últimos anos, é visto um considerável crescimento em sua popularidade e utilidade, majoritariamente devido a computadores mais poderosos, conjuntos de dados maiores e técnicas para treinar redes mais profundas (GOODFELLOW; BENGIO; COURVILLE, 2016).

A aprendizagem profunda usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. A primeira camada em uma rede é chamada camada de entrada, enquanto a última é chamada camada de saída. Todas as camadas entre as duas são referidas como camadas ocultas. Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação (TALON, 2022).

Uma arquitetura de aprendizagem profunda é uma pilha multicamadas de módulos simples, todos (ou a maioria) sujeitos ao aprendizado, e muitos dos quais calculam mapeamentos de

entrada-saída não lineares. Cada módulo da pilha transforma sua entrada para aumentar tanto a seletividade quanto a invariância da representação (LECUN; BENGIO; HINTON, 2015).

Redes Neurais Recorrentes (do inglês, *Recurrent Neural Networks (RNN)*) é um tipo especial de rede neural artificial adaptada para trabalhar com dados de séries temporais ou dados que envolvem sequências. As redes neurais *feedforward* comuns destinam-se apenas a pontos de dados independentes entre si. No entanto, se tivermos dados em uma sequência de modo que um ponto de dados dependa do ponto de dados anterior, precisamos modificar a rede neural para incorporar as dependências entre esses pontos de dados. Embora seu objetivo principal seja aprender dependências de longo prazo, elas mostram dificuldades para aprender e armazenar informações por muito tempo. A RNN convencional possui o problema de quando é utilizada para treinar com sequências muito longas (KIM et al., 2016).

Para corrigir isso, uma ideia é aumentar a rede com uma memória explícita. A primeira proposta deste tipo é a de *Long Short Term Memory(LSTM)* que usam unidades ocultas especiais, e cujo comportamento é lembrar entradas por um longo tempo. As redes *LSTM* posteriormente provaram ser mais eficazes que as RNNs convencionais, especialmente quando possuem várias camadas para cada passo de tempo (LECUN; BENGIO; HINTON, 2015).

3.4.1 LSTM

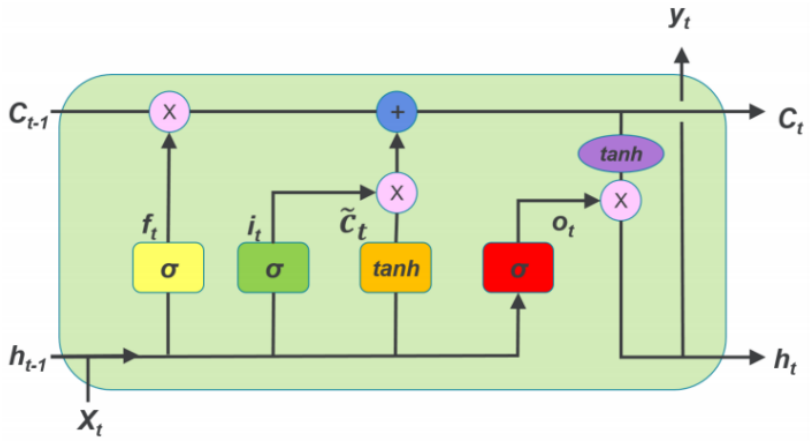
LSTM, é uma variação de RNN capaz de classificar, processar e prever séries temporais com intervalos de tempo indeterminado. A *LSTM*, tem um estado que atua como sua memória e internamente, esse estado decide o que manter, ou não, na memória (GOODFELLOW; BENGIO; COURVILLE, 2016).

Esses tipos de estados são muito eficientes na captura de dependências de longo prazo, o que o torna menos popular e inútil em casos de entradas mais longas. No caso de sequências longas, ele não pode acompanhar as sequências de longo prazo, tendo em vista que o gradiente das células iniciais se torna aproximadamente zero, deixando essas células mortas e causando perda de informações.

Uma rede *LSTM*, tem um estado que atua como sua memória e internamente, esse estado decide o que manter, ou não, na memória, além disso, pode remover ou adicionar informações ao estado da célula através dos portões:

- **Portão do Esquecimento**: as informações que não são mais úteis no estado da célula

Figura 14 – Arquitetura *LSTM*, onde o X_t é o vetor de entrada, h_{t-1} é a saída da célula anterior, C_{t-1} é a memória da célula anterior, h_t consiste na saída da célula atual, C_t a memória da célula atual, i_t portão de entrada, f_t portão do esquecimento e o_t portão de saída.



Fonte: (OINKINA, 2015)

são removidas. Esta porta recebe 2 entradas: uma é a saída gerada pela célula anterior e outra é a entrada da célula atual. Após o viés e pesos necessários serem adicionados e multiplicados, a função *sigmóide* é aplicada ao valor. Se para um determinado estado de célula a saída for 0, a informação é esquecida e para a saída 1, a informação é retida para uso futuro.

- **Portão de Entrada**: responsável pela adição de informações úteis ao estado da célula. Primeiro, a informação é regulada usando a função *sigmóide* que filtra os valores a serem lembrados similarmente ao portão do esquecimento usando as entradas. Ele cria uma matriz de informações que devem ser adicionadas. Isso é feito usando outra função de ativação chamada *tanh* que gera valores entre -1 e 1.

- **Portão de Saída** : responsável pela tarefa de extrair informações úteis do estado da célula atual para ser apresentadas como uma saída. Primeiro, um vetor é gerado aplicando a função *tanh* na célula. Então, a informação é regulada usando a função *sigmóide*, que filtra os valores a serem lembrados e os valores do vetor e os valores regulados são multiplicados para serem enviados como uma saída e entrada para a próxima célula.

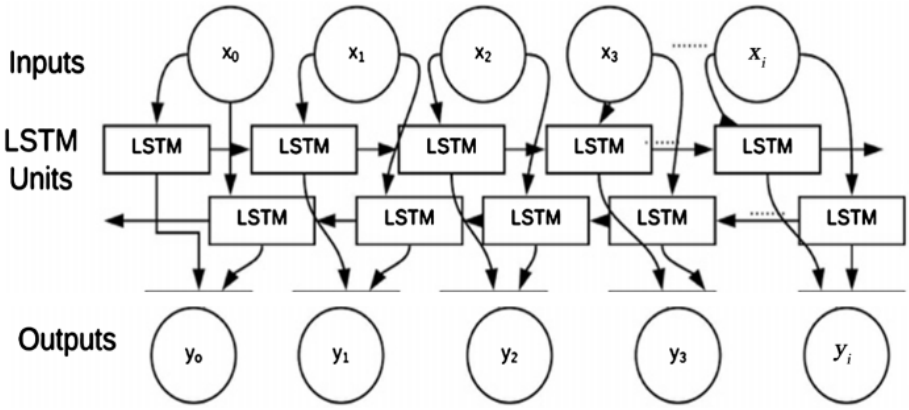
3.4.2 BiLSTM

As redes *BiLSTM* consistem em duas *LSTM* que funcionam em paralelo. Assim, quando é dada uma sequência de entrada, uma *LSTM* percorre tal sequência em uma direção - para

frente (*Forward LSTM*) - enquanto a outra *LSTM* vai na direção inversa - para trás (*Backward LSTM*). Uma estrutura bidirecional que permite ao vetor de estado oculto capturar informações passadas e futuras, dando maior poder de aprendizado para a rede (GRAVES; JAITLY; MOHAMED, 2013).

As redes *BiLSTM* aumentam efetivamente a quantidade de informações disponíveis, melhorando o contexto disponível para o algoritmo (por exemplo, saber quais palavras seguem e precedem imediatamente uma palavra em uma frase).

Figura 15 – As redes *LSTM* bidirecionais estendem as redes LSTM unidirecionais. Com o diferencial das duas redes *LSTM* trabalhando em paralelo, enquanto uma percorre em uma direção a outra *LSTM* vai na direção inversa, logo a rede consegue explorar informações do passado e do futuro.



Fonte: (KARAMITSOS; AFZULPURKAR; TRAFALIS, 2020)

3.5 PROCESSAMENTO DE LINGUAGEM NATURAL- PLN

Processamento de Linguagem Natural, do inglês, *Natural Language Processing(NLP)* investiga o uso de computadores para processar ou compreender linguagens humanas (ou seja, naturais) com o propósito de realizar tarefas úteis. O NLP é um campo interdisciplinar que combina linguística computacional, ciência, ciência cognitiva e inteligência artificial. Do ponto de vista científico, o NLP visa modelar os mecanismos cognitivos subjacentes à compreensão e produção de linguagem humana. Já no ponto de vista da engenharia, o NLP está preocupado em como desenvolver novas aplicações práticas para facilitar as interações entre computadores e linguagens humanas (DENG; LIU, 2018).

Algumas aplicações típicas em NLP incluem: sistemas de diálogo, análise textual, tradução automática, gráfico de conhecimento, recuperação de informações, resposta a perguntas, análise de sentimentos, computação social, geração de linguagem natural e resumo de linguagem

natural.

Geralmente, o processamento de linguagem natural utiliza algoritmos baseados em aprendizado de máquina, que conseguem aprender e compreender a linguagem humana a partir de dados de treinamento passados para o modelo. Existem várias tarefas de NLP como, por exemplo, tokenização, análise de dependência, remoção de *stopwords* e reconhecimento de entidades nomeadas (JACKSON; MOULINIER, 2007).

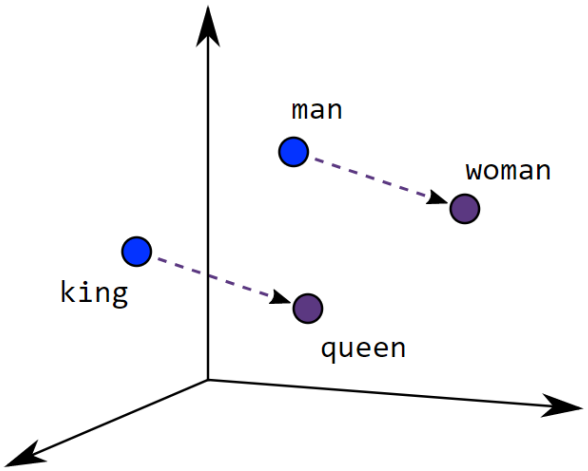
Este trabalho utiliza análise textual, tendo em vista que se baseia em informações textuais das requisições HTTP.

3.5.1 Representação Textual

As entradas passadas para os modelos de *deep learning* são vetores (matrizes de números), pois os modelos são incapazes de processar strings em sua forma bruta. Então, ao trabalhar com texto, a primeira tarefa é converter esse texto em números para alimentar o modelo, esse processo também é chamado de vetorizar o texto. Existem várias estratégias para realizar a vetorização dos textos, algumas delas são: codificações *one-hot*, onde o texto é representado por uma matriz de valores binários. Cada palavra é associada a um índice em um vetor com tamanho N, onde N é o tamanho do vocabulário(palavras sem repetições existentes no texto). O vetor é preenchido com 0, em todas as posições, com exceção de um único 1 no índice da palavra em questão.

Outra estratégia de vetorização é a de *word embeddings*, representações de palavras de forma vetorial, e basicamente, são técnicas de identificação de semelhanças entre palavras de um vocabulário, mapeadas em um espaço vetorial (CHOLLET, 2017). Isso é feito associando um vetor numérico a cada palavra em um dicionário, de forma que a distância entre quaisquer dois vetores captura parte da relação semântica entre as duas palavras associadas. Os *word embeddings* fornecem uma maneira de usar uma representação eficiente e densa, onde palavras semelhantes possuem uma codificação semelhante, enquanto os vetores obtidos com a codificação *one-hot* são esparsos. A Figura 16 mostra uma aplicação real de *word embeddings* em que os vetores representam a relação semântica entre gêneros.

Figura 16 – Representação de *word embeddings*



Fonte: (DEVELOPERS, 2022)

3.6 TRANSFORMERS

Google Brain publicou um artigo, ***Attention is all you need***, que introduziu uma nova arquitetura neural para NLP conhecida como: *Transformers*. O *Transformers*, é uma rede neural com arquitetura *encoder-decoder ou codificador-decodificador* baseada em um mecanismo de atenção que aprende as relações contextuais entre palavras em um texto. A rede recebe uma sequência de palavras como entrada, codifica-as em representações nas camadas de atenção e as decodifica em palavras novamente (VASWANI et al., 2017).

A arquitetura *Transformers* conforme Figura 17 gera uma sequência a partir de outra iterativamente. Com esses dados, o modelo produz uma distribuição de probabilidades para o próximo elemento na sequência de saída. O elemento com a maior probabilidade é então adicionado à sequência de saída e o processo é repetido.

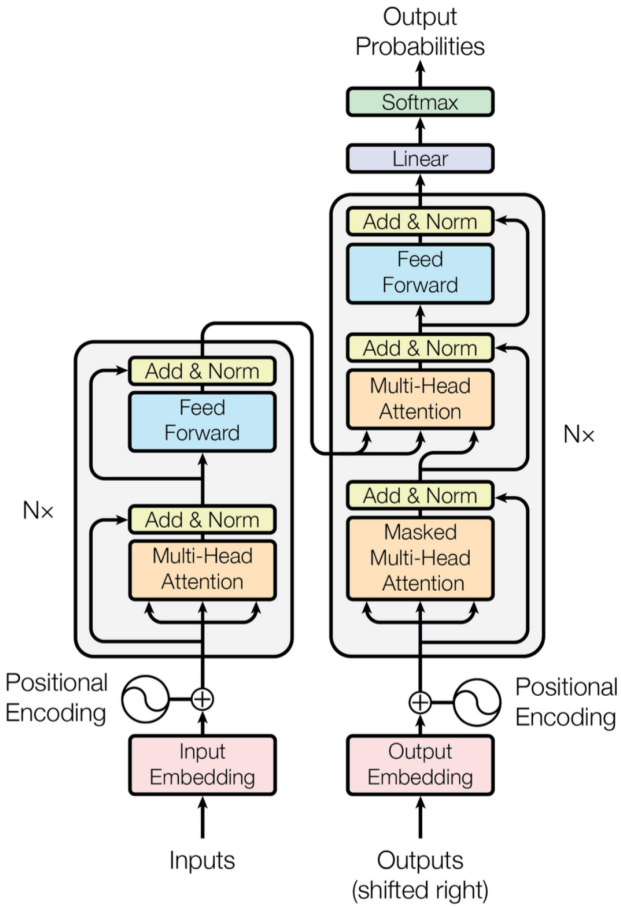
Antes de serem alimentadas ao modelo, as entradas do *Transformers* são primeiro convertidas em *embeddings*. Nesse procedimento, o conteúdo das sequências é codificado em forma vetorial. Contudo, as ordens dos elementos nessas sequências são perdidas. Com o propósito de resolver esse problema, os autores de (VASWANI et al., 2017) adicionaram uma codificação posicional que fornece informações sobre a posição de cada elemento da sequência, então se a mesma palavra aparecer em uma posição diferente, a representação real será um pouco diferente, dependendo de onde ela aparece na frase de entrada.

A arquitetura utiliza o mecanismo de atenção codificador-decodificador. O codificador está à esquerda e o decodificador à direita. Tanto o codificador quanto o decodificador são

compostos de módulos que podem ser empilhados um sobre o outro, várias vezes, o que é descrito por *Nx* na Figura 17. É possível visualizar que os módulos consistem principalmente em camadas *Multi-Head Attention e Feed Forward*.

Alguns modelos *Transformers* foram treinados como modelos de linguagem e em grandes quantidades de texto bruto de forma auto-supervisionada. Esses tipos de modelos desenvolvem uma compreensão estatística da linguagem em que foram treinados. Então, o modelo geral pré-treinado passa por um processo chamado aprendizagem por transferência. Durante esse processo, o modelo é ajustado de forma supervisionada em uma determinada tarefa. Esse trabalho utiliza o modelo de linguagem *BERT*, explicado na Seção 3.6.1.

Figura 17 – *Transformers* é uma arquitetura codificador-decodificador. O codificador consiste em um conjunto de camadas de codificação que processa a entrada iterativamente uma camada após a outra e o decodificador consiste em um conjunto de camadas de decodificação que fazem a mesma coisa com a saída do codificador.



Fonte: (VASWANI et al., 2017)

3.6.1 Bidirectional Encoder Representations from Transformers (BERT)

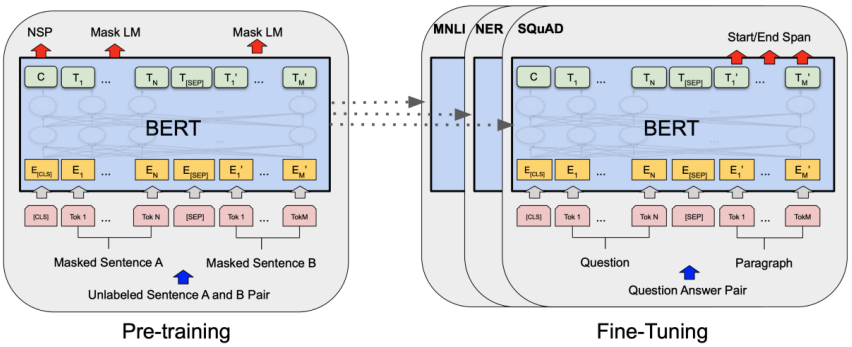
É um modelo de linguagem baseado na arquitetura *Transformers* (VASWANI et al., 2017) que alcançou o estado da arte em onze tarefas de NLP (DEVLIN et al., 2018). Ao contrário dos modelos recentes de representação de linguagem, o *Bidirectional Encoder Representations from Transformers (BERT)* é projetado para pré-treinar representações bidirecionais profundas de texto não rotulados, condicionando conjuntamente em ambos os contextos esquerdo e direito em todas as camadas. Como resultado, o modelo *BERT* pré-treinado pode ser ajustado com apenas uma camada de saída adicional para criar modelos de última geração para uma ampla série de tarefas, como responder a perguntas e inferência de linguagem, sem modificações substanciais na arquitetura específica da tarefa (DEVLIN et al., 2018).

A concepção por trás do *BERT* pode ser separada em duas etapas. Na primeira, o modelo é treinado exaustivamente para conseguir compreender uma língua. Para isso, ele recorre ao codificador e decodificador, os quais têm a tarefa tradicional de um *autoencoder*: conseguir representar o input da rede (no caso, requisições HTTP) em um espaço multidimensional complexo que, nesse caso, tem a propriedade de capturar as particularidades da língua necessárias à sua compreensão. Para atingir esse objetivo, o modelo é treinado com um conjunto robusto de textos, preferencialmente de natureza variada, para que de fato ele consiga compreender a língua. Essa etapa é definida como a de pré-treinamento, como ilustrado na Figura 18 (CECCON, 2020).

Os codificadores e decodificadores do *BERT* possuem em sua estrutura interna implementações dos *Transformers*, que têm a função de capturar as inter-relações entre as diferentes partes do texto que estão sendo processadas. Nessa etapa de modelagem da linguagem, o *BERT* é treinado não para reconstruir o texto original, mas com dois outros objetivos. O primeiro é prever algumas palavras mascaradas aleatoriamente. O segundo é prever se uma sentença é sequência lógica da sentença anterior. Assim, ele é forçado a usar as palavras na proximidade da palavra mascarada e o contexto das duas sentenças apresentadas ao modelo em sequência, para então realizar a predição correspondente e, dessa forma, aprender a inferir uma palavra através do contexto da sentença no qual está inserida, assim como entender as relações entre duas sentenças sequenciais.

Na segunda etapa de sua aplicação, depois que o *BERT* já foi treinado para entender uma língua, ele pode ser usado nas tarefas mais diversas. Esse é o processo clássico de *transfer learning*, e trata da transferência do aprendizado efetuado na primeira etapa para uma tarefa

Figura 18 – Procedimentos gerais de pré-treinamento e *fine-tuning* para *BERT*. Na fase de pré-treinamento o *BERT* pode aprender uma representação geral da linguagem por meio de duas tarefas bem projetadas: modelo de linguagem mascarado (MLM) e previsão da próxima sentença (NSP). Já para o *fine-tuning* pode ser ajustada para diversas tarefas *downstream* por meio de treinamento supervisionado em conjuntos de dados rotulados e para uma tarefa específica, basta anexar uma camada personalizada na última camada do *BERT*.



Fonte: (DEVLIN et al., 2018)

específica. Para isso, o modelo usa a parte do codificador, com seus parâmetros ajustados na tarefa anterior, e depois, passa essa representação do texto para um classificador. No caso do *BERT*, esse classificador costuma ser uma rede neural densa simples, sem nenhuma camada escondida, mas nada impede que estruturas mais complexas sejam usadas. E a essa segunda etapa tem-se como a de (*fine-tunning*).

O treinamento de um modelo BERT envolve duas etapas: o pré-treinamento e o *fine-tuning*, a Figura 18 mostra uma representação de cada uma das etapas, à esquerda, a etapa de pré-treinamento *(pre-training)* e à direita, o *fine-tuning*. A primeira etapa é feita em duas tarefas: *Masked Language Model (Mask LM) e Next Sentence Prediction (NSP)*.

A aplicação do *BERT* não está limitada, entretanto, à classificação de textos ele pode também ser usado em tarefas mais complexas.

3.6.1.1 Masked LM

Antes de alimentar as sequências de palavras no *BERT*, 15% das palavras em cada sequência são substituídas por um token *[MASK]*. O modelo então tenta prever o valor original das palavras mascaradas com base no contexto fornecido pelas outras palavras não mascaradas da sequência.

Na prática, a implementação do *BERT* não substitui os 15% de todas as palavras mascaradas. Para mitigar isso, quando um token é escolhido para ser mascarado, 80% das vezes o token é substituído pelo token *[MASK]*, 10% das vezes é substituído por um token aleatório do corpus,

e 10% do tempo permanece o mesmo (GUO; YUAN; WU, 2021).

A função de perda *BERT* considera apenas a previsão dos valores mascarados e ignora a previsão das palavras não mascaradas. Como consequência, o modelo converge mais lentamente do que os modelos direcionais, uma característica compensada por sua maior consciência de contexto.

3.6.1.2 Next Sentence Prediction (NSP)

No processo de treinamento do *BERT*, o modelo recebe pares de sentenças como entrada e aprende a prever se a segunda sentença do par é a sentença subsequente no documento original. Durante o treinamento, 50% das entradas são um par no qual a segunda sentença é a subsequente no documento original, enquanto nos outros 50% uma sentença aleatória do corpus é escolhida como segunda sentença. A suposição é que a sentença aleatória será desconectada da primeira sentença.

Para discriminar duas sentenças, um token especial *[SEP]* é colocado ao final de cada frase. Além disso, um token *[CLS]* é inserido no início da sentença. Após incorporar esses tokens especiais na sequência de entrada, uma incorporação posicional é adicionada a cada token para indicar sua posição na sequência.

3.7 TÉCNICAS E MODELOS COMPARATIVOS

Nesta seção serão mostrados técnicas e alguns modelos que serão utilizados como linhas de base para confrontar com o modelo proposto nesta dissertação. Cada modelo possui uma técnica diferente e todos serão testados com os mesmos conjuntos de dados que o RequestBERT-BiLSTM.

3.7.1 Word2Vec

Word2Vec é um método estatístico que realiza eficientemente um *Word Embedding* independente, a partir de um *corpus* de texto (MIKOLOV et al., 2013). Além disso, a eficácia do *Word2Vec* vem de sua capacidade de agrupar vetores de palavras semelhantes. (MIKOLOV et al., 2013)

O *Word2Vec* foi utilizado com a finalidade de expor a utilização do *word embedding*

tradicional.

3.7.2 NeuralLog

O modelo *NeuralLog* (LE; ZHANG, 2021) tenta superar as limitações das abordagens existentes, aplicando uma nova abordagem de detecção de anomalias baseada em log que usa diretamente mensagens de log brutas para detectar anomalias (LE; ZHANG, 2021).

No geral, *NeuralLog* consiste em três etapas: pré-processamento, representação neural, e classificação baseada em *Transformers*. O primeiro passo é o pré-processamento de log. Depois disso, cada mensagem de log é codificada em um vetor semântico usando *BERT* e por fim classificadas.

Em resumo, o *NeuralLog* tem como propósito detectar anomalias em logs sem a realização de *Log Parsing*. Os autores de (LE; ZHANG, 2021) relatam haver perda de informações ao ser feita o *parsing* para cada log. No entanto, no seu projeto há uma função que realiza o tratamento dos logs que retira todos os números e caracteres especiais, ficando apenas letras. Todavia, isso pode acarretar problemas, pois exclui informações que podem ser sensíveis.

3.7.3 CNN

O modelo CNN (LU et al., 2018) foi um dos primeiros trabalhos a serem criados para explorar a viabilidade da rede CNN para detecção de anomalias baseada em log (CHEN et al., 2021).

Os autores construíram um parser que estrutura as linhas de log e cria uma chave única para cada log. Nessas chaves são excluídas as informações menos utilizadas (LU et al., 2018). Esse *parser* de log foi construído aplicando particionamento baseado em janela, onde o preenchimento ou truncamento é aplicado para obter comprimentos de sequência consistentes.

3.7.4 Modelos BERT

Foram criados modelos que utilizam *BERT* com abordagens diferentes visando comparação com o modelo proposto neste trabalho. No primeiro cenário foi utilizado o *BERT* padrão sem adição de nenhuma outra classe na etapa de *fine-tunning*, já outro cenário foi a adição de uma rede CNN *BERT-CNN*.

3.7.5 Considerações

A ideia de utilizar os diferentes modelos foi tentar abordar diferentes perspectivas, onde será possível a comparação com o modelo proposto nesta dissertação. Sendo assim, foi abordado um cenário com *word embedding* tradicional com *Word2Vec* visando verificar como esta solução se comportaria para o cenário. A vantagem do *embedding* tradicional é a facilidade de encontrar corpus pré-treinado com facilidade, no entanto, eles são mais rígidos quanto ao contexto. Outra ideia foi utilizar o mesmo *embedding* tradicional adicionando uma rede Convolucional com objetivo de verificar o uso de *deep learning* com *embedding* tradicional.

Já para os cenários com o *BERT* foi para mostrar as diversas formas que pode ser utilizado-o. Primeiramente tem-se o *NeuralLog* que utiliza *BERT* em seu projeto com a camada densa na saída do *BERT*. O *BERT* padrão utiliza o modelo pré-treinado *BertForSequenceClassification* com a mesma camada densa na saída do *BERT*, logo o objetivo desse modelo foi verificar como seria o comportamento do modelo sem quaisquer ajustes. Agora no caso do *BERT*-CNN foi a ideia do *BERT* com a adição de uma rede CNN com o propósito de verificar o comportamento de uma camada CNN e *BERT* juntas.

A vantagem de utilizar o *BERT* é trazer para o cenário de detecção de ataques em requisições HTTP a capacidade de compreender a relações entre palavras e frase e reconhecer o contexto completo de cada palavra utilizada. Identificando e considerando tanto os termos que vêm antes quanto os que vêm depois de cada vocábulo.

4 TRABALHOS RELACIONADOS

Este capítulo tem por objetivo trazer um esboço de outros estudos relacionados a essa pesquisa. Nesse sentido, procurou-se relacionar pesquisas sobre detecção de ataques em requisições HTTP e detecção de anomalias em logs para dar embasamento ao desenvolvimento dessa dissertação. Além disso, serão mostradas outras técnicas para detecção de ataques.

4.1 DETECÇÃO DE ATAQUES EM REQUISIÇÕES HTTP

Existem algumas técnicas de detecção de intrusão que utilizam aprendizagem de máquina para servidores Web com base em requisições HTTP. Em (YU et al., 2020) e (KUANG et al., 2019) ambos usam a linha de requisição.

Os trabalhos de (YU et al., 2020) e (KUANG et al., 2019) assumem que a grande parte dos ataques na Web são implementados pela manipulação do componente Path-URL, pois os experimentos consideraram apenas o conjunto de dados CSIC 2010 (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2009) que contém apenas ataques no campo Path-URL. No entanto, outros ataques também podem utilizar outros componentes da requisição HTTP, como os cabeçalhos HTTP, onde existem tipos de ataques como: *Session Hijacking and Cross-Site Request Forgery (CSRF)*, que utilizam o cookie de sessão do usuário na página.

O (YU et al., 2020) utiliza o conjunto de dados CSIC 2010 (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2009) e no seu pré-processamento: remove requisições duplicadas, converte cada caractere das requisições em minúsculas e decodifica os caracteres das requisições efetuadas por navegadores ou por invasores deliberadamente, ou seja, "%25"é convertido para "%", e por fim é feito o *embedding* das palavras com *Word2Vec* baseado em *Skip-Gram*.

Para o processamento do modelo, os autores de (YU et al., 2020) utilizam *TextCNN* (KIM, 2014) para extrair *features* automaticamente e transferir informações. A modelagem da rede CNN ficou como: 2 x 64, 3 x 64, 4 x 64 e 5 x 64.

Em (KUANG et al., 2019) criaram o *DeepWAF* composto por quatro módulos: analisador, pré-processador, detector e classificador. O processo típico para o *DeepWAF* detectar ataques numa requisição HTTP ocorre da seguinte forma: a requisição que chega no servidor Web é examinada pelo analisador através dos cabeçalhos HTTP. Em seguida, o pré-processador investiga a requisição HTTP e gera uma sequência da URL que será alimentada no detector.

Em seguida, o detector verifica se a requisição é normal ou maliciosa com base nos modelos de aprendizado profundo integrados. Finalmente, o classificador classifica a requisiação. O *DeepWAF* somente extrai das requisições HTTP o campo URL para os métodos *GET* e *POST*. No entanto, para os casos de requisições que contenham *POST*, também é tratado o campo *body* da requisição.

Com relação aos modelos adotados para classificação, os autores tentaram com *CNN, LSTM, LSTM-CNN e CNN-LSTM*. A estrutura da rede para CNN foi: 3 x 128, 4 x 128, 5 x 128 e 6 x 128 e para o *LSTM* foram definidas 64 o número de unidades ocultas.

O modelo proposto, *CLCNN*, por (ITO; IYATOMI, 2018) foi construído com o propósito de realizar a identificação em requisições HTTP maliciosas assumindo a implementação prática de um WAF. O sistema executa uma rede CNN ao nível de caracteres baseado na requisição HTTP, ou seja, o sistema recebe o texto completo da requisição HTTP como entrada e após transfere para a rede CNN especializada em processamento de texto em cadeias de caracteres numa direção unidimensional. Cada letra da requisição HTTP é convertida em uma expressão vetorial de 128 dimensões na camada de *embedding* e propagada para as camadas subsequentes.

O trabalho de (LIU et al., 2022) esboça um modelo composto por quatro módulos. Os dois primeiros módulos visam a extração de recursos eficientemente. No primeiro módulo, a requisição é extraída e rotulada através de um processo de pré-processamento. Em seguida, a sequência de blocos é construída pelo bloco deslizante para remover informações, os itens de alta frequência na sequência de blocos são selecionados por um dicionário. No processo de blocos de *embedding*, os recursos baseados em blocos são construídos codificando cada item em sequência de blocos em um vetor de *embedding*. Os dois últimos módulos formam a última parte da estrutura proposta, que visa detectar anomalias de forma adaptativa para a requisição HTTP.

4.2 DETECÇÃO DE ATAQUES BASEADAS EM URL

Outra abordagem é extrair *features* das URLs como em: (ALTHUBITI; YUAN; ESTERLINE, 2017), (RONG; ZHANG; LV, 2018) e (XUAN; DINH; VICTOR, 2020) que consiste basicamente em selecionar os atributos mais relevantes extraídos das URLs contidas nas requisições.

Em (ALTHUBITI; YUAN; ESTERLINE, 2017) conseguiram selecionar nove *features* para o processo de detecção em requisições HTTP. Os autores utilizaram os métodos de seleção de *features* no *Weka* como: método avaliador de atributo e método de busca. O avaliador de

atributos é uma técnica que mostra como cada atributo no conjunto de dados é avaliado no contexto da saída, enquanto o método de pesquisa representa como os atributos podem ser navegados ou explorados no conjunto de dados. Além disso, foi utilizado os avaliadores de atributos: *WrapperSubsetEva* e o *BestFirst* como método de busca para navegar pelos subconjuntos de atributos. Alguns dos atributos referem-se ao comprimento das requisições, o tamanho dos argumentos, o comprimento do path da url ou o cabeçalhos.

Em seguida, foram utilizados alguns modelos de *machine learning(random forest, logistic regression, AdaBoost, J48, SGD e Naïve Bayes)* para testar.

Já em (RONG; ZHANG; LV, 2018), testaram o *embedding* ao nível de caracteres somente das palavras contidas na *query string* da URL, conforme a Figura 19. O sistema inclui cinco módulos que são: **pré-processamento de dados, modelo de classificação, verificação manual, banco de dados rotulado e modelo de treinamento**. E o sistema funciona da seguinte forma: O módulo de pré-processamento de dados recebe a requisição HTTP e extrai o parâmetro de requisição depois convertendo-a em uma lista de índices. Em seguida, esse módulo alimenta esta lista de índices no modelo de detecção. Após alguns cálculos sofisticados com relação a essa lista de índices, o modelo de detecção retorna o resultado da detecção.

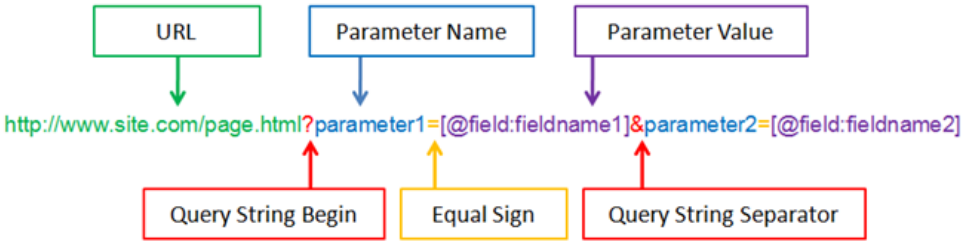
No background do sistema em (RONG; ZHANG; LV, 2018), o modelo de classificação salva algumas amostras e seus resultados de detecção. Essas amostras salvas são selecionadas aleatoriamente daquelas amostras sobre as quais o modelo de detecção considera menos confiável. Para garantir que os dados rotulados sejam representativos e que a quantidade de cada tipo de dados é balanceada, é necessário a verificação das amostras manualmente antes de adicioná-los ao banco de dados rotulado. Quando o sistema descobre que o banco de dados rotulado mudou, ele treina um novo modelo de detecção baseado no modelo antigo usando o banco de dados alterado.

Em (XUAN; DINH; VICTOR, 2020) os autores extraem as *features* lexicais e agrupam características específicas dos hosts contidas nas URL. Os atributos das URLs são extraídos com base em comportamentos estáticos e dinâmicos. Alguns grupos de atributos são tratados incluindo caracteres e grupos semânticos.

Os autores utilizaram alguns grupos de extração de *features* para a detecção de URL maliciosa. Os métodos utilizados de ***features* lexicais** com: comprimento das URLs, comprimento do domínio, comprimento máximo do domínio do token, média do path url; ***features* baseados em host** com: características de host nas URLs(localização e identidade) e ***features* baseados em conteúdo** com: a captura da página Web. Posteriormente, foram utilizados *Randon Forest*

e SVM para classificação.

Figura 19 – Exemplo de *Query String*. É um conjunto de pares/valores anexados a URL. Após a URL de determinada página, é adicionado o primeiro valor usando a seguinte sintaxe: "? Chave=Valor". Para passarmos mais de um conjunto, os mesmos devem ser concatenados usando o caractere coringa &.



Fonte: (CASPIO, 2022)

Zolotukhin et al. (ZOLOTUKHIN et al., 2014) considerou que as requisições HTTP podem ser analisados para detecção de intrusão a partir de acessos normais. Para isso eles executaram os treinamentos baseados nas requisições HTTP que não contém nenhum ataque e extraíram toda informação relevantes das requisições, onde técnicas de agrupamento e detecção de anomalias foram aplicadas para definir um modelo de comportamento normal das requisições e realizavam comparações com as requisições suspeitas.

4.3 DETECÇÃO DE ANOMALIAS EM OUTROS TIPOS DE LOGS

Quanto a detectar anomalias em logs, em geral, podemos destacar alguns trabalhos da literatura.

Lu et al. (LU et al., 2018) construiu um analisador de log para examinar e filtrar os logs brutos e transformá-los em dados estruturados formados por chaves de log, para então excluir informações inúteis como *timestamp* de registros específicos. Em seguida, é codificada cada chave de log analisada com um número único. Especificamente, são contadas quantas chaves de log exclusivas o conjunto de dados possui e associa cada chave de log exclusiva em um número único. Por fim, este analisador de log aplica um particionamento baseado em janela para reagrupar essas chaves de log para diferentes sessões. Após classificar essas chaves de log em ordem, tem-se uma sessão estruturada. Assim, cada sessão inclui um identificador exclusivo e uma série de chaves de log relacionadas como uma sessão de log em cada bloco. Posteriormente, os autores criam um processo de *embedding*: *logkey2vec*, que é uma matriz treinável cuja forma é igual a: quantidade de chaves de logs únicos x 128 para mapear cada

chave de log em uma sessão num vetor (CHEN et al., 2021). Depois disso, a incorporação é inserida em uma rede *CNN*: com 3 camadas *CNN* em paralelo com tamanhos 3 x 128, 4 x 128 e 5 x 128.

Du et al. (DU et al., 2017) propôs o *DeepLog*, com três principais componentes: o modelo de detecção de anomalia de chave de log, o modelo de detecção de anomalia de valor de parâmetro e o modelo de fluxo de trabalho para diagnosticar anomalias detectadas. Na fase de treinamento os dados para *DeepLog* são entradas de log de execução normal do sistema. Cada entrada de log é transformada em uma chave de log e um valor no vetor de parâmetros. A sequência de chaves de log analisada do arquivo de log de treinamento é usado pelo *DeepLog* para treinar um modelo de detecção de anomalia de chave de log e para construir modelos de fluxo de trabalho de execução do sistema para fins de diagnóstico. Na fase de detecção, o *DeepLog* primeiro usa a chave de log modelo de detecção de anomalias para verificar se a chave de log de entrada é normal. Se sim, o *DeepLog* verifica ainda mais o vetor de valor do parâmetro usando o modelo de detecção de anomalia de valor de parâmetro para essa chave de log. A nova entrada será rotulada como uma anomalia se sua chave de log ou seu vetor de parâmetro for classificado como anormal. Por último, se o log é rotulado como anormal, o modelo de fluxo de trabalho do *DeepLog* fornece informações semânticas para os usuários diagnosticarem a anomalia.

DeepLog (DU et al., 2017) foi primeiro a trabalhar com o emprego de LSTM para detecção de anomalias de log. Em suma, os padrões de log são aprendidos a partir das relações sequenciais de eventos de log, onde cada mensagem de log é representada pelo índice de seu evento de log.

V. Le et al. (LE; ZHANG, 2021) propôs o NeuralLog que consiste em três etapas: pré-processamento, representação neural e classificação baseada em *Transformers*. A primeira etapa é o pré-processamento de log. Depois disso, cada mensagem de log é codificada em um vetor semântico usando *BERT* e *WordPiece* para capturar o significado de palavras no nível de subpalavra. Além disso, como o modelo de classificação é baseado em *Transformers*, logo melhora o desempenho de detecção de anomalias. Em resumo, os autores visaram detectar anomalias em logs sem realizar um *Log Parser* combinando com *BERT* e realizando uma adaptação de janela para criar modelos de log.

Zhang et al. (ZHANG et al., 2019) propuseram o *LogRobust* e observaram que muitos estudos existentes de detecção de anomalias de log falham em alcançar o desempenho prometido na prática. Particularmente, a maioria deles assumem: 1) os dados de log são estáveis ao longo do tempo; 2) os dados de treinamento e teste compartilham um conjunto idêntico de eventos

de log distintos (CHEN et al., 2021). A extração das informações semânticas de eventos de log aproveitando vetores de palavras prontos para uso, é um dos primeiros estudos a considerar a semântica dos logs. O *LogRobust* incorpora a *attention mechanism* (VASWANI et al., 2017) em um modelo *BiLSTM* para atribuir pesos diferentes para registrar eventos, chamado *attentional BiLSTM*.

4.4 CONSIDERAÇÕES

Conforme a busca e avaliação dos trabalhos relacionados realizada, foi possível observar que os trabalhos acima apresentaram avanços significativos, mas poucos exploraram profundamente técnicas de *deep learnning* em requisições HTTP. No entanto, as técnicas de *deep learnning* são abordagens promissoras para aprender com dados de amostra legítimos ou maliciosos sem a necessidade de análise de log.

Além disso, é possível notar que em diversos trabalhos da literatura nenhum utiliza o *BERT* para detecção de ataques em requisições HTTP, apenas para outros cenários de detecção de ataque. Diante deste contexto, o RequestBERT-BiLSTM trará evolução para este panorama com o emprego do *BERT* com uma camada *BiLSTM* que proporcionará a eficiência na compreensão das requisições, possibilitando a detecção de anomalias.

5 REQUESTBERT-BILSTM

Neste Capítulo, será apresentada uma proposta que visa a detecção de ataques em requisições HTTP combinando *BERT* e *BiLSTM*. O objetivo principal do RequestBERT-BiLSTM é aprender as informações contidas nas requisições HTTP e conseguir detectar ataques.

As requisições HTTP são mensagens enviadas pelo cliente para iniciar uma ação no servidor. O trabalho de detecção é baseado nas informações contidas nos campos *HTTP Request*.

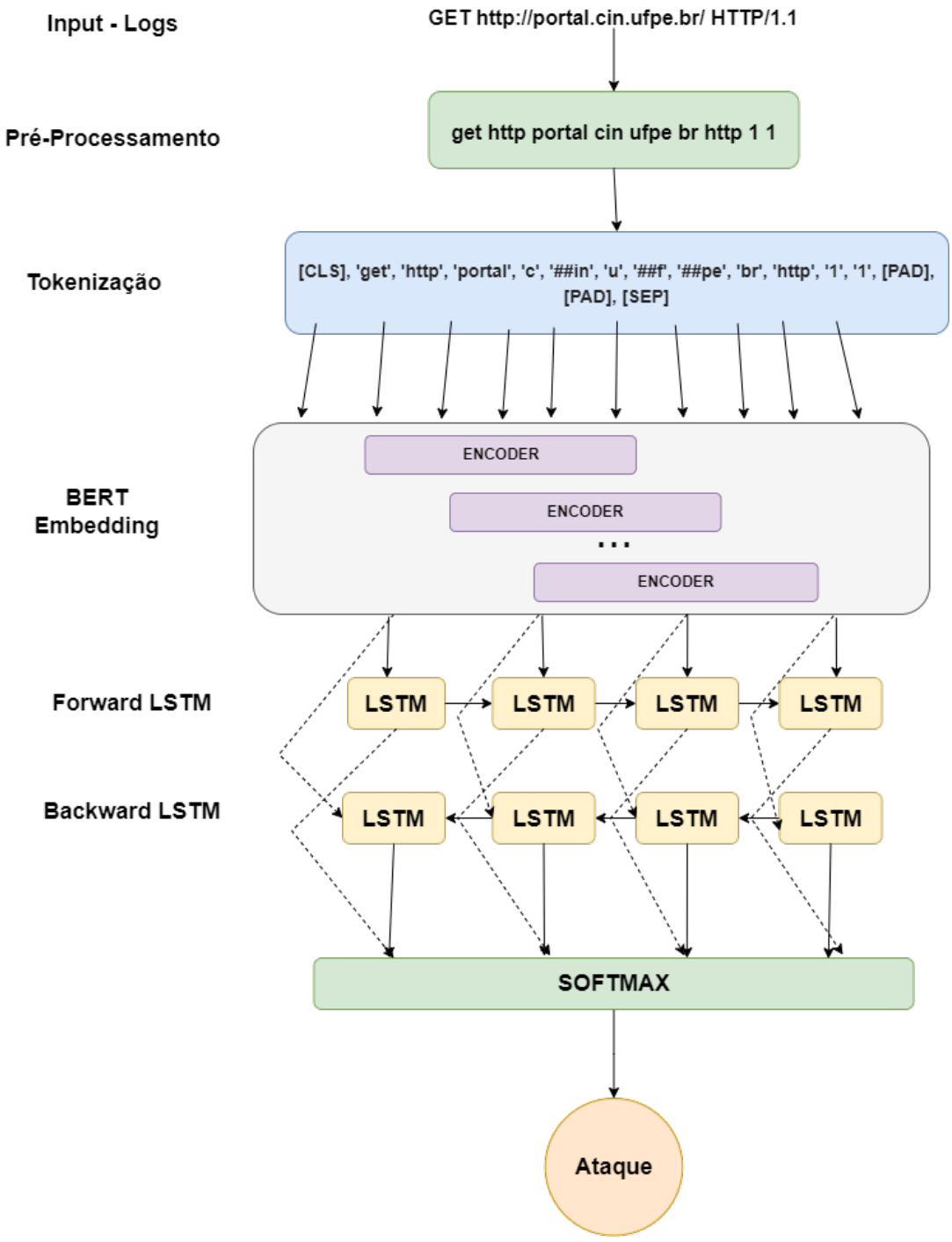
RequestBERT-BiLSTM é um algoritmo supervisionado baseado em *BERT* e *BiLSTM*. A escolha do *BERT* é com objetivo de utilizá-lo como parte de entrada de dados e *BiLSTM* como parte de classificação do modelo. Com a rede *BiLSTM* é possível fazer uso eficiente de recursos passados (através de estados avançados) e recursos futuros (através de estados anteriores) por um período específico, tornando-se eficiente em aprender as sequências de informações nas duas direções. O propósito é construir um modelo combinando essas duas abordagens, e o resultado é a possibilidade de prever possíveis ataques nas requisições HTTP.

O RequestBERT-BiLSTM foi desenvolvido para requisições HTTP e utiliza todo conteúdo para extrair a semântica das requisições HTTP. Além disso, ele implementa o aprendizado por transferência com a utilização do *BERT*, possibilitando reciclar o conhecimento do modelo previamente treinado, para então proporcionar o treinamento apenas da nova camada *BiLSTM* inserida no modelo proposto neste trabalho.

Outro ponto que pode ser destacado para o RequestBERT-BiLSTM é não ser restrito em requisições HTTP, sendo também capaz de detectar anomalias em outras categorias de logs, como será mostrado posteriormente.

A Figura 20 mostra a arquitetura de uma forma geral do RequestBERT-BiLSTM. Cada etapa da arquitetura está muito bem definida, projetada e operando da seguinte forma: primeiro acontece o pré-processamento das requisições HTTP, extração de contexto pelo *BERT* e o processo de bidirecionalidade do *BiLSTM*, por fim o modelo rotula se a requisição HTTP é ataque ou não. A classificação dos ataques depende do conjunto de treinamento e são necessárias no mínimo 10 entradas para o modelo conseguir compreender a categoria do ataque.

Figura 20 – Arquitetura do RequestBERT-BiLSTM, em que é possível observar os passos até a detecção do ataque nas requisições HTTP. A entrada de dados no modelo são as requisições HTTP, a próxima etapa é um tratamento simples nos dados, deixando-os minúsculos e retirando caracteres especiais. O processo seguinte de tokenização do *BERTTokenizer*, separa as palavras em sub-palavras com objetivo que continuem sendo representadas no modelo. Nesta etapa há inserções de tokens como *[CLS], [PAD] e [SEP]* que servem, respectivamente, para indicar o início da sentença, preenchê-las para chegarem ao tamanho do token definido e para indicar o término da sentença. Já na etapa de *BERT Embedding*, após receber uma sequência de no máximo 512 tokens é possível gerar a representação da sequência. O próximo passo é o diferencial do RequestBERT-BiLSTM, pois acrescenta uma rede *BiLSTM*.



Fonte: Elaborado pelo autor, 2022.

5.1 PRÉ-PROCESSAMENTO

Etapa inicial do projeto, onde as requisições são adaptados para o modelo. Inicialmente, como as requisições são extraídas em formato texto, foi necessária a criação de um script Python para convertê-las para CSV com a separação em colunas a partir de um Regex[1]. Um exemplo de uma linha do arquivo de log é mostrada na Figura 21, no entanto, alguns campos foram ocultados, pois são informações sensíveis e somente o campo *Request* é retirado do log. Portanto, após a conversão para CSV ficou mais simples a manipulação das informações.

O passo seguinte, já com o arquivo CSV separados em colunas, foi transformá-las em minúsculo e remover todos os caracteres especiais. Estes caracteres especiais removidos contém sinais de acentuação, pontuação e matemáticos.

A Figura 22 mostra uma requisição e como fica após o pré-processamento de remover caracteres especiais e a transformação em minúsculos na Figura 23.

Figura 21 – Log original capturado no ativo de segurança.

```
Dec 19 16:03:38 bigip.5cta.eb.mil.br ASM: SrcIp: 45.155.126.222 SrcPort:
    39687 DstIp: 192.168.10.197 DstPort: 80 User: N/A SupportId:
    5302773078426082833 SignId: N/A AttackType: Other Application Activity
     Uri: / Request: GET / HTTP/1.1\r\nHost: 177.8.95.72\r\nUser-Agent:
    Mozilla/5.0 (compatible; tchelebi/1.0; +http://tchelebi.io)\r\nAccept:
     */*\r\nAccept-Encoding: gzip\r\nX-Forwarded-For: 45.155.126.222\r\n\r
    \n
```

Fonte: Elaborado pelo autor, 2022.

[1]Regex é a abreviação do inglês *Regular Expressions*, para expressões regulares. Com regex é possível buscar, validar e alterar qualquer padrão de caracteres em qualquer texto.

Figura 22 – Apenas o campo *Request* foi retirado do log original conforme a Figura 21. Uma requisição HTTP de um possível ataque XSS ao servidor web.

```
GET
/sell-media-search/?keyword=%22%3E%3Cscript%3Ealert%281337%29%3C%2Fscript
   %3E HTTP/1.1\r\n
Host: sismil.7rm.eb.mil.br\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Connection: close\r\n
Accept: */*\r\n
Accept-Language: en\r\n
Accept-Encoding: gzip\r\n
X-Forwarded-For: 45.155.205.137\r\n\r\n
```

Fonte: Elaborado pelo autor, 2022.

Figura 23 – Requisição HTTP após o tratamento inicial: colocando em minúsculas e retirando os caracteres especiais.

```
get
sell media search keyword 22 3e 3cscript 3ealert 281337 29 3c 2fscript 3e
    http 1 1 r n
host sismil 7rm eb mil br r n
user agent mozilla 5 0 windows nt 10 0 win64 x64 r n
connection close r n
accept r n
accept language en r n
accept encoding gzip r n
x forwarded for 45 155 205 137 r n r n
```

Fonte: Elaborado pelo autor, 2022.

Figura 24 – Divisão da palavra em várias subpalavras, após a tokenização.

```
['get', 'sell', 'media', 'search', 'key','##word', '22', '3', '##e', '3',
'##cs', '##cript', '3', '##eal', '##ert', '281', '##33', '##7', '29',
'3', '##c', '2', '##fs', '##cript', '3','##e', 'http', '1', '1', 'r', 'n'
, '##hos','##t', 'si', '##sm', '##il', '7', '##rm', 'e','##b', 'mi',... '
   n']
```

Fonte: Elaborado pelo autor, 2022.

Figura 25 – IDs exclusivos das palavras ou subpalavras.

```
[101, 1243, 4582, 2394, 3403, 2501, 12565, 1659, 124, 1162, 124, 6063,
   13590, 124, 13003, 7340, 25567, 23493, 1559, 1853, 124, 1665, 123,
   22816, 13590, 124, 1162, 8413, 122, 122, 187, 183, 15342, 1204, 27466,
    6602, 2723, 128, 9019, 174, 1830, 1940, 1233, 9304, 187, 183, 102,...
    ]
```

Fonte: Elaborado pelo autor, 2022.

Figura 26 – Preenchimento com padding [PAD].

```
[101, 1243, 4582, 2394, 3403, 2501, 12565, 1659, 124, 1162, 124, 6063,
   13590, 124, 13003, 7340, 25567, 23493, 1559, 1853, 124, 1665, 123,
   22816, 13590, 124, 1162, 8413, 122, 122, 187, 183, 15342, 1204, 27466,
    6602, 2723, 128, 9019, 174, 1830, 1940, 1233, 9304, 187, 183, 102,...
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Fonte: Elaborado pelo autor, 2022.

Figura 27 – Tensor binário que indica a posição dos índices preenchidos com [MASK].

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
   1, 1, 1,.... 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Fonte: Elaborado pelo autor, 2022.

5.2 TOKENIZAÇÃO

É responsável por preparar as entradas para o modelo. A Tokenização divide strings em tokens de subpalavra e converte os tokens em ids e vice-versa. Também tem como atribuição a adição de tokens especiais (como máscara, início de frase, etc.) adicionando-os, atribuindo-os no *tokenizer* para facilitar o acesso e garantindo que não sejam divididos durante a tokenização.

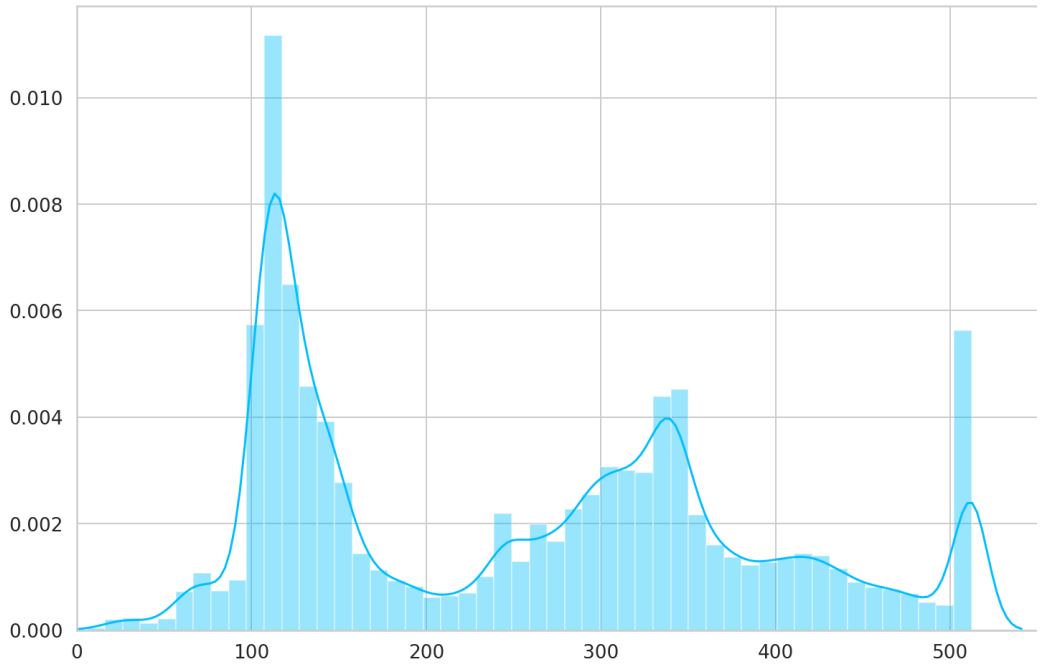
A representação do primeiro token é feita pelo token adicional *[CLS]*, sendo adicionado à sentença de entrada. Na tarefa de previsão da próxima frase, há a necessidade de informar ao *BERT* onde termina a primeira frase e onde começa a segunda frase. Assim, outro token adicional, *[SEP]*, é introduzido, como é possível observar na Figura 20 na etapa de Tokenização.

Há um ponto importante a ser observado quando é usado um modelo pré-treinado. Como o modelo é pré-treinado em um determinado *corpus*, o vocabulário também foi corrigido. Em outras palavras, quando se utiliza um modelo pré-treinado, é possível que alguns tokens nos novos dados não apareçam no vocabulário fixo do modelo pré-treinado. Diante deste cenário, seria o caso de eles serem substituídos por um token especial *[UNK]*, que significa token desconhecido. No entanto, converter todos os tokens não vistos em tokens desconhecidos removeria muitas informações dos dados de entrada. Assim, o *BERT* recorre a um algoritmo *WordPiece* que divide uma palavra em várias subpalavras, de modo que as subpalavras comumente vistas também possam ser representadas pelo modelo, conforme a Figura 24. Na Figura 24 é possível visualizar os casos das palavras divididas em subpalavras para manter o contexto, este é o exemplo de tokenização da requisição da Figura 22.

Como o modelo do *BERT* é pré-treinado, cada token recebe um ID exclusivo. Portanto, é necessário converter cada token do log de entrada em seus IDs exclusivos correspondentes conforme é mostrado na Figura 25. Onde cada palavra ou subpalavra possui um ID. A Figura 25 mostra os ID correspondentes as palavras ou subpalavras da Figura 24, onde os IDs 101 e 102 indicam *[CLS] e [SEP]*.

Devido à restrição do modelo *BERT* receber um comprimento fixo de sentença como entrada é necessário que no caso dos logs menores seja adicionado o preenchimento (token vazio) às sentenças para compor o comprimento máximo. Esse token é o *[PAD]*, ele é usado para representar os preenchimentos e normalmente para isso é utilizado o valor 0 para realizar esse preenchimento conforme a Figura 26. Tratando-se do comprimento máximo, no RequestBERT-BiLSTM este valor é definido após a visualização da distribuição dos tamanhos das requisições como mostra a Figura 28 e o valor escolhido é próximo ao tamanho máximo. No caso mostrado

Figura 28 – Gráfico mostrando o comportamento do tamanho das requisições do Dataset Log-Security.



Fonte: Elaborado pelo autor, 2022.

na Figura 28 o valor foi 400.

Já para o último cenário no processamento das requisições no *BERT* a Figura 27 mostra que *Attention Mask* é um tensor binário que indica a posição dos índices preenchidos com token *[MASK]*. Para o *BertTokenizer*, 1 indica um valor que deve ser atendido, enquanto 0 indica o valor de *[PAD]* a ser preenchido.

5.3 TRANSFORMER-BASED CLASSIFICATION

Neste trabalho, é utilizado o modelo pré-treinado *BERT-base (uncased)* que contém um *encoder* com 12 camadas de *transformers*, 12 *self-attention heads* e 768 camadas ocultas e não faz distinção entre maiúscula e minúscula. A escolha por este modelo em detrimento ao *BERT- large* que contém 24 camadas de *transformers*, 16 *self-attention heads* e tamanho 1024 camadas ocultas, foram os resultados iniciais já obtidos com o *BERT-base*.

A biblioteca *Transformers* possui a classe *BertForSequenceClassification* projetada para tarefas de classificação. No entanto, no RequestBERT-BiLSTM foi criada uma camada *BiLSTM* para ser possível a etapa de classificação de maneira mais robusta.

5.4 DETECÇÃO DE ATAQUES

A ideia de aplicar o RequestBERT-BiLSTM para detecção de ataques é que ele possa alcançar alta precisão de previsão em possíveis ataques em requisições HTTP e em outros logs de uma forma geral. No próximo Capítulo será mostrada a versatilidade que o RequestBERT-BiLSTM tem em prever ataques e detectar anomalias em logs.

6 EXPERIMENTOS

Neste Capítulo serão mostradas ordenadamente cada uma das etapas dos experimentos realizados na pesquisa. Sendo assim, a seção 6.1 apresenta os modelos utilizados para comparar com o RequestBERT-BiLSTM, a seção 6.2 apresenta os métodos de análise de log com suas configurações e um exemplo para elucidar o funcionamento de cada método, a seção 6.3 discorre sobre os conjuntos de dados utilizados e a criação do conjunto de dados deste trabalho com todas as suas propriedades e particularidades. Por fim, a seção 6.4 discorre sobre as métricas utilizadas para avaliação dos modelos.

O RequestBERT-BiLSTM foi avaliado em quatro conjuntos de dados, utilizou o otimizador *AdamW* com a taxa de aprendizado em $2e{-}5$, o tamanho do *batch* em 16 e com o máximo de 5 épocas. O modelo adotado neste trabalho foi desenvolvido em *Pytorch* no *GoogleColab*, utilizando GPU Tesla T4 com 16GB. A escolha do otimizador e a taxa de aprendizado foi baseado nos resultados iniciais obtidos, já o tamanho do *batch* e a quantidade de épocas foram escolhidos conforme o trabalho de (DEVLIN et al., 2018), em conjunto com os resultados.

6.1 MODELOS AVALIADOS

Para provar especificamente as vantagens da combinação do RequestBERT-BiLSTM na detecção de ataques em requisições HTTP, foram utilizados alguns modelos com abordagens diferentes sendo testados nos mesmos conjuntos de dados. A explicação e motivos da escolha desses modelos foram com objetivo de exibir técnicas utilizadas na comunidade cientifica, abordagens tradicionais e modelos com arquiteturas semelhantes, cujas propostas experimentadas sejam as mais próximas de cenários reais e de soluções já implementadas.

O modelo Word2Vec, é para mostrar a utilização do *word embedding* da maneira tradicional. Para esse modelo foi utilizado a base pré-treinada *Word2Vec* do *Google News* com 3 bilhões de palavras, ou seja, 3 milhões de vetores de palavras em inglês de 300 dimensões.

Após a inclusão da base pré-treinada, o *Word2Vec* realiza a correlação dos *embedding* contidos na base do *Google News* com as palavras do dataset. Após a criação da matriz de *embedding*, ela será utilizada como dados de entrada para o *KNN (k-nearest neighbors algorithm) ou algoritmo de k-vizinhos mais próximos* para testar a eficiência do modelo. Com isso, para todos os datasets foi utilizado o KNN com 3 vizinhos sem o *tunning* de

hiperparâmetros.

Para o modelo *CNN*, foi utilizada uma rede Convolucional (LU et al., 2018) sem *BERT* e utilizando *Fasttext* para a realização do *word embedding*. Então, para realizar a convolução que requer uma entrada de recurso bidimensional, os autores propuseram um método de incorporação chamado *logkey2vec*, que é um tipo de *embedding* no formato da quantidade de logs únicos x 128. Esse método realiza a conversão de vetor no nível do modelo de log e as informações semânticas no nível da palavra são ignoradas. Esse tratamento difere da incorporação de palavras que usa palavra como unidade de granulação fina, como *Word2Vec* (CHEN et al., 2021).

A próxima etapa foi a criação da rede CNN, e conforme os seus experimentos a estrutura da rede *CNN* ficou com: 3 camadas CNN em paralelo com tamanhos 3 x 128, 4 x 128 e 5 x 128. A função de ativação adotada foi a *Leaky Rectified Linear Unit (leaky ReLU ou LReLU)*, por fim, criou-se uma camada *max-pooling* para concatenar as camadas CNN.(LU et al., 2018)

O modelo padrão do *BERT* foi empregado apenas com a biblioteca *BertForSequence-Classification* que é composta apenas com a uma camada densa simples. O *BERT* utiliza o otimizador *AdamW* com a taxa de aprendizado em $2e{-}5$, o *batch-size* com 16 e o máximo de 5 épocas.

Já o modelo *BERT-CNN* fica compreendido entre o *BERT* padrão e o RequestBERT-BiLSTM, pois ele foi criado para mostrar um modelo utilizando *BERT* que possui uma classe distinta a *BertForSequenceClassification* e a *BiLSTM*. Nesse caso foi criado uma rede Convolucional seguindo os mesmo parâmetros de (LU et al., 2018), ficando com 3 camadas *CNN*: 3 x 128, 4 x 128 e 5 x 128.

Os valores de hiperparâmetros utilizados nos modelos propostos que utilizam *BERT* são os mesmos que o RequestBERT-BiLSTM.

6.2 MÉTODOS DE ANALISADORES DE LOGS

Baseado nas escolhas dos métodos de analisadores de log definidos na Seção 3.3.1 é possível comprovar que a opção de utilização de análise de log deve ser cuidadosa. Vale ressaltar que, os analisadores foram testados somente no conjunto de dados, Log-Security.

Para cada analisador a forma de execução foi semelhante. Como o objetivo deste trabalho é tratar toda a requisição HTTP não foi utilizado nenhum método de expressão regular para extrair qualquer informação específica. Cada método exige para sua execução, dois scripts

Python. O primeiro é para receber o arquivo texto com as requisições sem as classificações e qualquer pré-tratamento. Já o segundo script é responsável em produzir os 2 arquivos CSV, no qual um desses arquivos mostra a quantidade e tipos de templates gerados pelo analisador e o outro CSV mostra as requisições tratadas baseadas nos templates.

No caso do analisador *Drain* (HE et al., 2017) os valores dos parâmetros ajustáveis foram: *st=0.5 (Threshold de similaridade)* e *depth = 4 (profundidade da árvore)*.

Para o analisador *AEL* (JIANG et al., 2008b) os valores foram: *minEventCount = 2 (mínimo de eventos por grupo)* e *merge_percent = 0.5 (porcentagem de tokens diferentes em cada requisição)*.

Já para o *LFA* (NAGAPPAN; VOUK, 2010) não houve a necessidade de especificar nenhum parâmetro. No *LogSig* (TANG; LI; PERNG, 2011) a única informação a ser passada foi a quantidade de cluster, sendo assim o valor escolhido foi 16 por ser a mesma quantidade de classes do dataset Log-Security.

Para exemplificar, foi utilizado a requisição de um possível ataque *XSS* com um *SQL Injection* mostrada na Figura 29. Os templates provenientes dos analisadores *DRAIN, AEL, LogSig e LFA* baseados nos seus domínios podem ser observados, respectivamente, nas Figuras 30, 31, 32 e 33.

Com relação aos resultados dos *Log Parser* escolhidos é possível notar que, apenas no caso do *LogSig* - Figura 32 a informação sobre o ip de origem foi mantida, já nos outros casos é possível visualizar que as informações mais importantes foram ocultadas na formação dos templates.

No caso do *LFA* a distorção é muito grande. Essa dificuldade é possível observar na Figura 33, onde a quantidade de informações consideradas variáveis é grande.

Já para o caso do *LogSig* ele conseguiu extrair bem os dados e detectar as partes fixas e variáveis da requisição. O problema é que por considerar cluster e para o caso do exemplo houve diversas requisições do **ip: 45.155.205.137** ele entendeu que o ip seria informação fixa, no entanto, sabemos que não é, e o mesmo ocorre para navegador. Sendo assim, por mais que ele entenda que faça parte do template como informação fixa, elas certamente mudarão e consequentemente dados sensíveis serão desconsiderados.

Com relação aos analisadores, *Drain* e *AEL* que são da mesma categoria (heurística) tiveram divergência na compreensão para a mesma requisição HTTP. Um detalhe que chama a atenção é o *Drain* considerar o método HTTP como parte fixa, e os cabeçalhos: *Accept, Accpet-Language, Accept-Encodding e X-Forward-For*, já para o *AEL* essas mesmas informações

foram compreendidos de forma antagônica.

Desta forma, algumas palavras-chave essenciais nas requisições HTTP podem ser removidas após a análise de log, resultando em requisições diferentes. A requisição na Figura 29 foi compreendida de maneiras distintas por cada método de analisador de log. Sendo assim, esses erros degradarão o desempenho de qualquer modelo para detectar possíveis ataques em requisições HTTP. Além disso, ficou notório que a depender do analisador o resultado pode ser diferente.

Figura 29 – Requisição HTTP extraída do log original de um possível ataque.

```
GET /?server=db&username=root&db=mysql&table=event%3C%2Fscript%3E%3
    Cscript%3Ealert%28document.domain%29%3C%2Fscript%3E HTTP/1.1\r\nHost:
    sismil.7rm.eb.mil.br\r\nUser-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64
    ) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2226.0 Safari
    /537.36\r\nConnection: close\r\nAccept: */*\r\nAccept-Language: en\r\
    nAccept-Encoding: gzip\r\nX-Forwarded-For: 45.155.205.137\r\n\r\n
```

Fonte: Elaborado pelo autor, 2022.

Figura 30 – Template criado pelo analisador - **DRAIN**

```
GET <*> HTTP/1.1\r\nHost: <*> Mozilla/5.0 <*> <*> <*> <*> AppleWebKit
    /537.36 <*> like Gecko) <*> <*> close\r\nAccept: */*\r\nAccept-
    Language: en\r\nAccept-Encoding: gzip\r\nX-Forwarded-For: <*>
```

Fonte: Elaborado pelo autor, 2022.

Figura 31 – Template criado pelo analisador - **AEL**

```
<*> <*> HTTP/1.1\r\nHost: <*> Mozilla/5.0 <*> <*> <*> <*> AppleWebKit
    /537.36 <*> like Gecko) <*> <*> <*> <*> <*> <*> <*>
```

Fonte: Elaborado pelo autor, 2022.

Figura 32 – Template criado pelo analisador - **LogSig**

```
GET HTTP/1.1\r\nHost: Mozilla/5.0 (Windows NT AppleWebKit/537.36 (KHTML,
    like Gecko) Safari/537.36\r\nConnection: close\r\nAccept: */*\r\
    nAccept-Language: en\r\nAccept-Encoding: gzip\r\nX-Forwarded-For:
    45.155.205.137\r\n\r\n
```

Fonte: Elaborado pelo autor, 2022.

Figura 33 – Template criado pelo analisador - **LFA**

```
GET <*> HTTP/1.1\r\nHost: <*> Mozilla/5.0 <*> <*> <*> <*> <*> <*> <*> <*>
     <*> <*> <*> <*> <*> <*> <*>
```

Fonte: Elaborado pelo autor, 2022.

Os scripts dos métodos podem ser visualizados em: github.com/logpai/logparser.

6.3 DATASETS

Foram utilizados 4 dataset para o projeto, onde 3 são públicos: **ECML/PKDD 2007** (RAÏSSI et al., 2007), **CSIC 2010** (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2009), **BGL** (OLINER; STEARLEY, 2007) e por fim, um desenvolvido neste trabalho a partir do ativo de segurança F5 Big-IP: **Log-Security**.

Em 2007, a 18ª Conferência Europeia sobre Aprendizado de Máquina (ECML) e a 11ª Conferência Europeia sobre Princípios e Práticas de Descoberta de Conhecimento em Bancos de Dados (PKDD), apresentaram um desafio sobre Análise de Tráfego na Web.

O objetivo do desafio era construir um algoritmo baseado em técnicas de aprendizado de máquina para realizar classificação multiclasse e isolamento dos padrões de ataque. Como parte do desafio foi fornecido um conjunto de dados que continha tráfego válido e requisições classificadas em seis tipos diferentes de ataques: *Cross Site Scripting (XSS), SQL Injection, Injeção de LDAP, injeção de XPATH, Path Traversal e Command Execution*. O conjunto de dados contém 50.116 requisições, ficando com: 35.006 requisições classificadas como normais e 15.110 requisições classificadas como ataques. Ademais, o conjunto de dados é dividido em conjunto de treinamento, validação e teste com as respectivas quantidades: 30.069, 10.023 e 10.024. O conjunto de dados ECML/PKDD 2007 foi gerado registrando o tráfego real que foi então processado para sanitizar as informações. Esse processo de mascaramento consistia em renomear cada url, nomes de parâmetros e valores com strings geradas aleatoriamente (RAÏSSI et al., 2007).

O dataset CSIC 2010 foi desenvolvido pelo Instituto de Pesquisa em Segurança da Informação do Conselho Nacional de Pesquisa Espanhola (CSIC). O dataset contém os dados gerados de tráfego direcionado para um aplicativo Web de comércio eletrônico e inclui alguns ataques como: *XSS, SQL Injection, CRLF e Buffer Overflow*.

Nesta aplicação Web, os usuários podem comprar itens usando um carrinho de compras e se cadastrar fornecendo algumas informações pessoais. Por ser uma aplicação web em espanhol, o conjunto de dados contém alguns caracteres latinos. O conjunto de dados é gerado automaticamente e contém 36.000 requisições normais e mais de 25.000 requisições anômalas. As requisições HTTP são rotuladas como normais ou anômalas. O conjunto de dados é dividido em três subconjuntos diferentes: um subconjunto para a fase de treinamento, que possui apenas tráfego normal; e dois subconjuntos para a fase de teste, um com tráfego normal e outro com tráfego malicioso, resultando em 97 mil requisições.

O tráfego é gerado seguindo os seguintes passos:

Primeiro, os dados reais são coletados para todos os parâmetros das aplicações web. Todos os dados (nomes, sobrenomes, endereços, etc.) são extraídos de bancos de dados reais. Esses valores são armazenados em dois bancos de dados: um para os valores normais e outro para os anômalos. Além disso, todas as páginas públicas disponíveis das aplicações Web são listadas. Em seguida, requisições normais e anômalas são geradas para cada página web. Caso as requisições normais tenham parâmetros, os valores dos parâmetros são preenchidos com dados retirados do banco de dados normal aleatoriamente. O processo é análogo para requisições anômalas, no qual os valores dos parâmetros são obtidos do banco de dados anômalo (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2009).

Foram considerados três tipos de requisições anômalas:

1) **Ataques estáticos que tentam solicitar recursos ocultos (ou inexistentes)**. Essas requisições incluem arquivos obsoletos, ID de sessão na reescrita de URL, arquivos de configuração, arquivo padrão, etc.

2) **Ataques dinâmicos que modificam argumentos de requisições válidas**: *XSS, SQL Injection, CRLF e Buffer Overflow* etc.

3) **Requisições ilegais não intencionais**. Essas requisições não têm intenção maliciosa, porém não seguem o comportamento normal da aplicação Web e não possuem a mesma estrutura dos valores normais (por exemplo, um número de telefone composto por letras).

Os ataques foram gerados com a ajuda de ferramentas como *Paros e W3AF*.

Os WAFs em que este conjunto de dados foi usado seguem a abordagem de anomalia, ou seja, o comportamento normal da aplicação Web é padrão e o comportamento diferente é considerado anômalo. Portanto, nesta abordagem apenas o tráfego normal é necessário para a fase de treinamento.

Para a realização do treinamento no modelo RequestBERT-BiLSTM tanto o conjunto de dados, ECML e o CSIC foram incorporados para ficar em um único arquivo e sem separação de treinamento e teste, pois em cenários reais não há essa separação há somente um grande conjunto de dados que mistura requisições anômalas e normais.

O dataset BGL é um conjunto de dados aberto de logs coletados de um sistema de supercomputadores *BlueGene/L no Lawrence Livermore National Labs (LLNL)* em Livermore, Califórnia, com 131.072 processadores e 32.768 GB memória (OLINER; STEARLEY, 2007). O log contém mensagens de alerta e não alerta identificadas por tags de categoria de alerta. Na primeira coluna do log, '-' indica mensagens sem alerta, enquanto outras são mensagens de

alerta (HE et al., 2020).

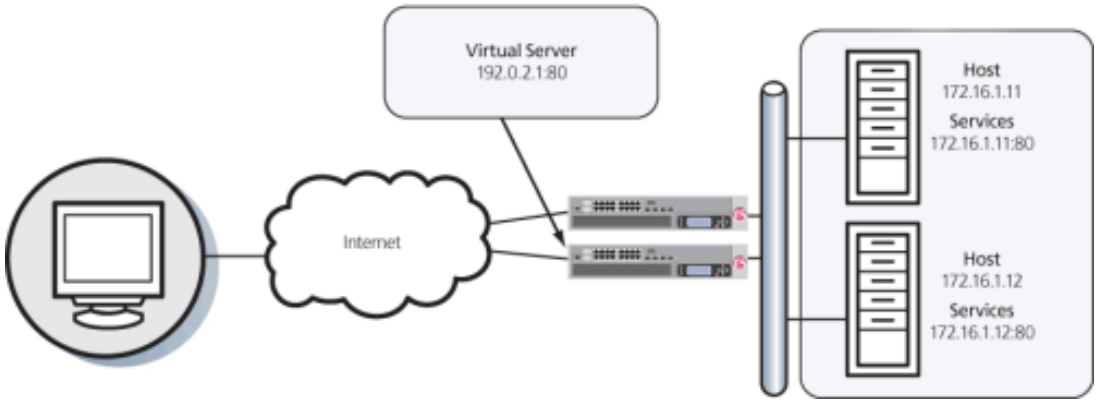
Tabela 1 – Tabela dos Datasets

Dataset	Categoria	Tamanho	Classes	Anomalias
BGL	Supercomputador	4.7 milhões	Binária	348.460
Log-Security	Request HTTP	98 mil	Multi	67265
CSIC 2010	Request HTTP	97 mil	Binária	25000
ECML/PKDD	Request HTTP	50 mil	Multi	15110

6.3.1 Dataset - Log Security

O conjunto de dados criado nesse trabalho foi extraído a partir do ativo de segurança, *F5 Big-IP*. O *F5 Big-IP* é um *firewall* de rede *full-proxy, stateful* e de alto desempenho projetado para proteger *datacenters* contra ameaças recebidas que entram na rede através dos protocolos, *HTTP/S, SMTP, DNS e FTP*.

Figura 34 – Uma das principais funções do sistema BIG-IP é direcionar diferentes tipos de protocolo e tráfego de aplicativos para um servidor de destino apropriado.



Fonte: (F5BIG-IP, 2022)

F5 Big-IP mostrado na Figura 34 pode encaminhar o tráfego diretamente para um pool de servidores de balanceamento de carga ou enviar o tráfego para um roteador. Na Figura 34 mostra a posição que o *F5 Big-IP* deve ser colocado, ficando à frente dos servidores e recebendo diretamente as requisições da internet.

Inicialmente o *F5 Big-IP* foi configurado para filtrar somente as requisições consideradas ofensivas, ou seja, essas requisições já estariam rotuladas baseado no conhecimento de assinaturas do equipamento. Todas as requisições do conjunto de dados são destinadas a sites

hospedados na infraestrutura de um provedor real. Além disso, algumas dessas requisições foram criadas propositalmente através do sistema *KALI Linux*. [1] e suas ferramentas. As ferramentas do KALI utilizadas para realizar as tentativas foram:

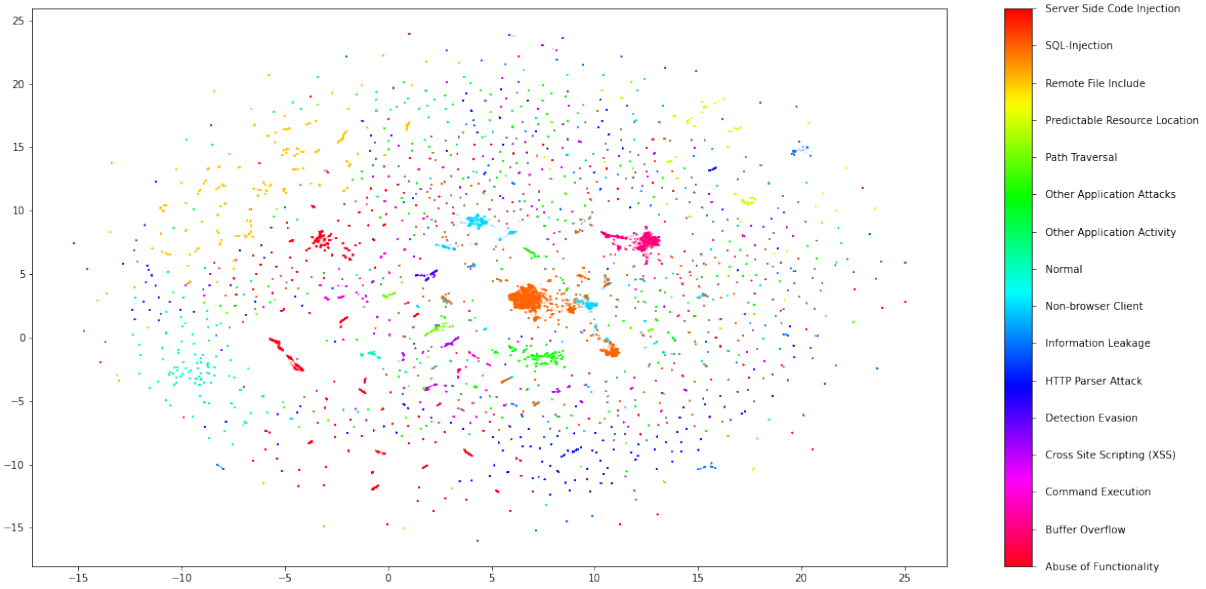
- **BeEF**: é usado para explorar uma vulnerabilidade de XSS e se concentra em ataques do lado do cliente. Uma vez que a ferramenta explora o XSS em um site, os usuários desse site se tornam a vítima e seu navegador pode ser totalmente controlado pelo BeEF;
- **BetterCap**: realiza o Main-the-middle visando manipular o tráfego HTTP, HTTPS e TCP em tempo real. Além disso, o BetterCap consegue quebrar SSL / TLS pré-carregado;
- **Network Mapper (Nmap)**: ferramenta de scanner de rede que permite escanear um sistema ou uma rede. O Nmap permite que você escaneie portas abertas e serviços em execução;
- **THC Hydra**: é um cracker de senha que realiza o bypass de login de rede, utilizando dicionário ou ataque de força bruta para tentar várias combinações de senha e login. Ele pode executar ataques rápidos de dicionário contra mais de 50 protocolos; e
- **SQLMap**: o objetivo do SQLMap é detectar e aproveitar as vulnerabilidades de injeção de SQL em servidores Web. Após detectar uma ou mais brechas nas injeções de SQL no servidor, o atacante pode escolher uma variedade de opções: listar usuários, hashes de senha, privilégios, realizar o dump das tabelas/colunas, executar sua própria instrução SQL, ler arquivos específicos no sistema de arquivos e muito mais.

Após as tentativas de intrusão e requisições diárias, foi realizado a configuração para que os logs do equipamento fossem automaticamente enviados a um centralizador de logs, com isso a manipulação pode ser efetuada sem a possibilidade de comprometer o funcionamento do *F5 Big-IP*. Após a coleta, foi realizado a adaptação do arquivo de log para CSV. Isso foi possível devido ao script Python que realiza toda a conversão e separação dos campos para colunas baseado em um Regex.

Para um melhor entendimento do dataset, Log-Security, criado neste projeto, foi necessário a projeções dos dados com *UMAP*. A Figura 35 mostra alguns clusters formados e mais ao

[1] Kali Linux é uma distribuição Linux de código aberto baseada em Debian voltada para várias tarefas de segurança da informação, como teste de penetração, pesquisa de segurança, computação forense e engenharia reversa. <https://www.kali.org/>

Figura 35 – Representação do conjunto de dados Log-Security com *UMAP*. Com relação aos clusters formadas mais ao centro ficam evidenciadas 3 categorias de ataques em que conseguimos destacar algumas características: para o SQL_Injection há a necessidade de alguma palavra-chave como *DML e DDL - SELECT, INSERT, UPDATE e DELETE e ALTER, CREATE e DROP*. No caso de Comand Execution existe há mesma necessidade de comandos específicos como - *cat, run* etc. e por fim, nos casos de *Buffer Overflow* o destaque é o tamanho de suas requisições, logo esses ataques possuem padrões bem definidos. Também é possível notar pequenos grupos das classes Normal e *Other Application Attacks*, refletindo diversas requisições com características semelhantes, no entanto, o normal é que essas duas classes não possuam nenhuma padronização.



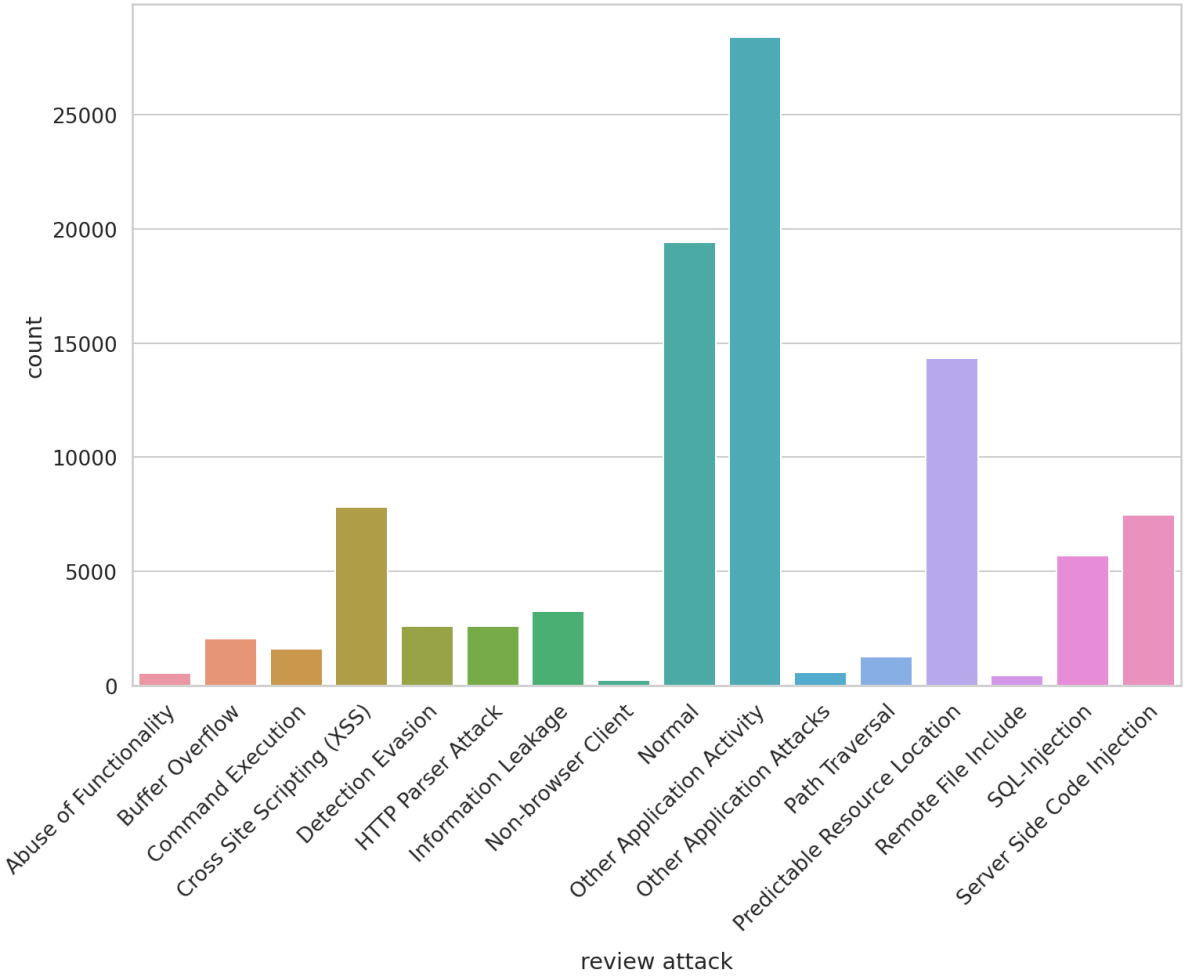
Fonte: Elaborado pelo autor, 2022.

centro é possível visualizá-los e identificá-los como ataques: *SQL_Injection, Comand Execution e Buffer Overflow*.

A Figura 36 mostra o quanto esse dataset é desbalanceado de fato retratando um cenário real. Segundo a Figura 36 há mais requisições suspeitas, pois o objetivo foi justamente colher a maior quantidade possível de requisições maliciosas para o modelo aprender o comportamento das requisições maliciosas. E isso é possível visualizar também na Tabela 1, onde mais da metade das requisições são de ataques.

A escolha pelo dataset BGL foi para mostrar a versatilidade que o RequestBERT-BiLSTM tem em poder ser utilizado em qualquer cenário que trate de anomalias em logs.

Figura 36 – Distribuição das classes no Dataset Log-Security. O dataset contém acessos normais e acessos rotulados com os respectivos ataques como: *Abuse of Functionality, Authentication/Authorization Attacks, Buffer Overflow, Command Execution, Cross Site Scripting (XSS), Detection Evasion, HTTP Parser Attack, Information Leakage, Non-browser Client, Other Application Activity, Other Application Attacks, Path Traversal, Predictable Resource Location, Remote File Include, SQL-Injection, Server Side Code Injection* e *Vulnerability Scan*. Com relação ao tipo de ataque *Other Application Attacks* representa ataques que não se enquadram nas classificações de ataque mais explícitas (F5-ATTACK, 2022). E a Figura 36 mostra o quanto esse dataset é desbalanceado, de fato retratando um cenário real.



Fonte: Elaborado pelo autor, 2022.

6.4 MÉTRICAS UTILIZADAS

Para mensurar a performance do RequestBERT-BiLSTM na detecção de ataques em requisições HTTP foram utilizados as seguintes métricas: **Precisão, Recall, F1-Score, Acurácia, ROC e AUC**. Os cálculos e a definição das métricas são descritos abaixo:

- *True Positive* (TP): Indica a quantidade de registros classificados como positivos corretamente;
- *True Negative* (TN): Indica a quantidade de registros classificados como negativos de

maneira correta;

- *False Positive* (FP): Indica a quantidade de registros classificados como comentários positivos de maneira incorreta;
- *False Negative* (FN): Indica a quantidade de registros classificados como comentários negativos de maneira incorreta.

6.4.1 Precisão

Precisão refere-se à porcentagem dos resultados dos sistemas reconhecidos corretamente. É a razão entre os termos identificados corretamente (*True Positive*) e a soma dos termos *True Positive* e *False Positive*

$$Precisao = \frac{TP}{TP + FP}$$

6.4.2 *Recall*

Refere-se à porcentagem do total de entidades corretamente reconhecidas pelo sistema. Indica com que frequência o classificador está encontrando exemplos de uma classe. É a razão entre os termos *True Positive* e a soma dos termos *True Positive* e *False Negative*, definida;

$$Recall = \frac{TP}{TP + FN}$$

6.4.3 *F1-Score*

Média harmônica das métricas Precisão e *Recall*, definida pela Equação 2.3. Essa métrica combina a Precisão e o *Recall* de modo a trazer um número único que indique a qualidade geral do modelo e trabalha bem até com conjuntos de dados que possuem classes desproporcionais;

$$F1 = \frac{2 \times Precisao \times Recall}{Precisao + Recall}$$

6.4.4 Acurácia

É a quantidade de acertos do nosso modelo divido pelo total da amostra;

$$Acuracia = \frac{TP + TN}{Total}$$

6.4.5 ROC e AUC

Receiver Operating Characteristic (ROC) é uma curva de probabilidade, ela é criada traçando a taxa verdadeiro-positivo contra a taxa de falsos-positivos, ou seja, o número de vezes que o classificador acertou a predição contra o número de vezes que o classificador errou a predição. Assim, na tentativa de simplificar a análise da ROC, a *Area Under the ROC Curve (AUC)* nada mais é que uma maneira de resumir a curva ROC em um único valor, agregando todos os limiares da ROC, calculando a "área sob a curva". E a AUC representa o grau ou medida de separabilidade. Quanto maior o AUC, melhor o modelo está em prever 0s como 0s e 1s como 1s.

7 RESULTADOS E DISCUSSÃO

Este Capítulo apresenta os resultados obtidos na execução dos experimentos.

7.1 COMPARATIVOS ENTRE OS MÉTODOS DE LOG PARSER

A Tabela 2 mostra a quantidade de templates criados e tempo de duração de cada analisador para o mesmo dataset, Log-Security. Os parâmetros de cada método está na Seção 6.2.

O método que levou menor tempo foi *LFA*, já o maior foi o *LogSig*. Tratando-se de métodos para a mesma categoria, o *DRAIN* e *AEL* divergiram bastante com relação ao tempo de execução, porém para quantidade de template ficou próximo. O tempo do *LogSig* se dá pelo fato das tentativas de verificação de semelhança entre as linhas de requisição para cada cluster.

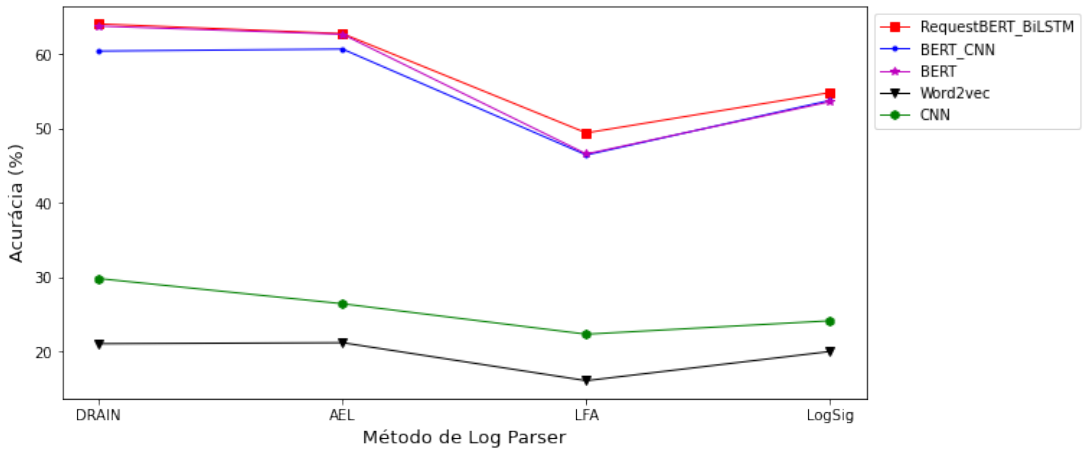
Tabela 2 – Tabela de comparação dos Analisadores de Logs no dataset Log-Security

Log Parser	**Categoria**	**Nº Templates**	**Tempo**
DRAIN	Heurística	1781	1m3s
AEL	Heurística	1471	18m
LogSig	Clustering	16	3h37m
LFA	Padrões Frequentes	428	15s

A Figura 37 mostra o desempenho dos modelos tendo como entradas as requisições HTTP resultantes após a análise de log. Há um comportamento notável para maioria dos modelos: o método *DRAIN*, superando a maioria dos modelos testados, exceto no caso do *BERT-CNN* e *Word2Vec*, para esses dois modelos o *AEL* foi o melhor. Por outro lado, o *LFA* foi o pior em todos. Outro ponto que pode ser destacado é quanto os modelos que utilizam *BERT* ficaram próximos e, simultaneamente, muito superior aos outros.

Apesar da dificuldade encontrada e os valores de Acurácia com os dados gerados pelos métodos de analisador de log o RequestBERT-BiLSTM foi superior aos outros modelos. Os valores para todos os métodos para cada modelo estão na Tabela 3. Isso mostra que mesmo utilizando *Log Parser* o RequestBERT-BiLSTM foi melhor que os outros modelos experimentados.

Figura 37 – Resultados dos Métodos de *Log Parser* no Dataset Log-Security. A execução de todos os métodos de analisadores foi desenvolvida baseado nos valores de hiper-parâmetros conforme descrito na Seção 6.2 e na Tabela 3 é possível visualizar numericamente os resultados.



Fonte: Elaborado pelo autor, 2022.

Tabela 3 – Tabela de comparação dos analisadores de logs na base de dados, Log-Security.

Log Parser	RequestBERT-BiLSTM	BERT	BERT-CNN	Word2Vec	CNN
DRAIN	**64.08%**	63.79%	60.43%	21.06%	29.83%
AEL	**62.78%**	62.23%	60.70%	21.20%	26.44%
LFA	**49.42%**	46.59%	46.58%	16.09%	22.34%
LogSig	**54.85%**	53.62%	53.84%	20.02%	24.15%

7.2 COMPARATIVOS ENTRE OS MODELOS ANALISADOS

A Tabela 4 mostra todos os modelos testados neste projeto e resultados de outros trabalhos para os mesmos datasets.

7.2.1 RequestBERT-BiLSTM x Word2Vec

Neste cenário é notória a superioridade do RequestBERT-BiLSTM ao *Word2Vec*, como observado na Tabela 4, podendo essa superioridade ser explicada por diversos fatores. Primeiramente, a maneira de como acontece os *embedding* das palavras. O *Word2Vec* ele trata como formas de contexto baseadas em palavras, já o RequestBERT-BiLSTM utiliza tanto o *BERT* para o contexto de cada palavra, quanto a dupla abordagem que a camada *BiLSTM* implementa na saída do *BERT*. Sendo assim, para o propósito de detecção de ataques em requisições HTTP, o RequestBERT-BiLSTM foi muito superior ao *Word2Vec*.

7.2.2 RequestBERT-BiLSTM x CNN

Já com relação ao modelo proposto de *CNN* (LU et al., 2018), houve a dificuldade de adaptar o analisador de log desenvolvido no trabalho (LU et al., 2018). O analisador é baseado na quantidade de logs únicos por sessão e o dataset utilizado por eles foi *HDFS*, que é o conjunto de logs divididos, segundo os blocks_IDs por sessão. Além disso, o método do *logkey2vec* ignora as informações semânticas no nível da palavra, o que pode afetar a precisão dos resultados da detecção. Posto isso, como os datasets escolhidos para avaliação, nenhum se assemelha com o *HDFS*, a opção foi realizar o *word embedding* utilizando o *FastText*. [1].

A utilização do *FastText* foi feita com as mesmas medidas especificadas em (LU et al., 2018) e respeitando as configurações dos hiperparâmetros. Por fim, o RequestBERT-BiLSTM foi superior ao CNN, pois o processamento do *BERT* é mais eficiente que o *word embedding*.

7.2.3 RequestBERT-BiLSTM x NeuralLog

Ambos utilizam o *BERT* e não possuem *Log Parser* e uma das diferenças existentes entre eles está no tratamento inicial dos logs. O *NeuralLog* escolhe uma janela pré-determinada, e agrupa os logs dentro dessa janela para compor uma sequência de log. Além disso, há outro detalhe no momento da limpeza dos logs. O *NeuralLog* descarta os números, consequentemente, o *NeuralLog* tem um baixo desempenho. Já o RequestBERT-BiLSTM considera cada linha do log como uma entrada e não descarta os números, porque no panorama de detecção de ataques os números são relevantes e não devem ser desconsiderados.

As Figuras 22 e 29 mostram que os números guardam informações consideráveis, como o ip, versão do navegador entre outros.

Outro fator importante para o *NeuralLog* foi a necessidade de ajuste no código para executar de acordo com as características do dataset Log-Security, pois como o dataset é multiclasse o *NeuralLog* não executava, então foi necessário realizar a mudança no código e definir que só existiriam requisições legítimas e anômalas sem qualquer classificação, tornando a base de dados em binária. E mesmo com essa mudança o *NeuralLog* não conseguiu ter um desempenho satisfatório.

[1] Método para aprender representações de palavras, onde, cada palavra é representada por um conjunto de *ngrams* dos caracteres. Todo n-gram está associado com uma representação vetorial e em seguida, cada palavra é representada a partir da soma de todos os vetores dos seus *ngrams* (BOJANOWSKI et al., 2016)

Portanto, o *NeuralLog* teve problemas com relação à limpeza e o agrupamento de logs. E mesmo no cenário do conjunto de dados *BGL* que o *NeuralLog* foi desenvolvido, o RequestBERT-BiLSTM foi superior.

7.2.4 RequestBERT-BiLSTM x BERT

Os 2 modelos utilizam o *BERT* com as mesmas configurações e valores de hiperparâmetros. A diferença crucial está após a saída do *BERT*, pois para *BERT* ele utiliza a classe padrão *BertForSequenceClassification* que tem somente no topo uma camada densa com *dropout* e no RequestBERT-BiLSTM há a camada do *BiLSTM*. Essa diferença é que faz o desempenho do modelo proposto ser superior aos outros modelos que utilizam *BERT* em sua estrutura, tendo em vista que a vantagem que o *BiLSTM* entrega é bastante superior a uma camada densa.

7.2.5 RequestBERT-BiLSTM x BERT-CNN

O RequestBERT-BiLSTM x BERT-CNN é semelhante ao comparado com o *BERT* padrão, porém neste caso foi acrescentada rede neural Convolucional. Portanto, a conclusão que se pode ter é que a configuração da rede Convolucional utilizada por (LU et al., 2018) não é a ideal.

7.3 COMPARAÇÃO DE RESULTADOS

Na Tabela 4 existem modelos que tiveram dificuldades com as características dos conjuntos de dados. Sendo assim, após os experimentos esses modelos apenas apresentaram resultados para a classe majoritária, exibindo um valor elevado de Acurácia. Todas as ocorrências estão destacadas na Tabela 4 com a cor ciano.

Esse problema é conhecido como: **paradoxo de acurácia**, ou seja, os algoritmos não conseguem diferenciar a classe minoritária das demais categorias (UDDIN, 2019).

Essa falta de diferenciação pode ocasionar problemas sérios, uma vez que a identificação desses casos minoritários podem ser o cerne do desafio a ser resolvido. E como forma de evidenciar esse problema a métrica *AUC* foi utilizada, logo, os mesmos modelos que tiveram esse problema obtiveram aproximadamente 50% na métrica *AUC*, denotando que o modelo não tem capacidade de separação das classes, tornando-se completamente aleatório.

Tabela 4 – Comparação do Desempenho entre os Modelos e os Datasets

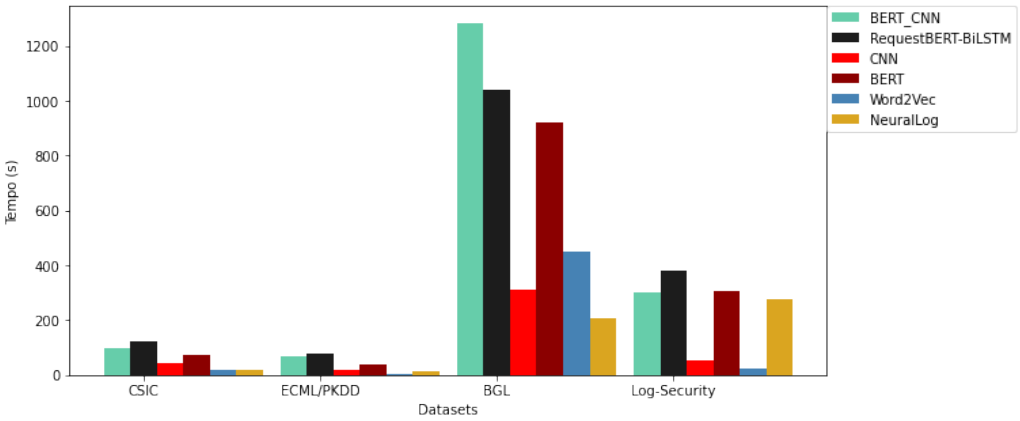
Dataset	Modelo	Recall (%)	Precisão (%)	F1-Score (%)	Acurácia (%)	AUC (%)
CSIC 2010	Word2Vec	63	62	62	62.66	49.55
	BERT	98	97	98	98.4	97.23
	BERT-CNN	98	98	99	99.06	99.17
	NeuralLog (LE; ZHANG, 2021)	95	94	96	97	50
	CNN (LU et al., 2018)	58	58	58	58.34	49.52
	CLCNN (ITO; IYATOMI, 2018)	n/a	n/a	n/a	98.8	n/a
	DeepWaf (KUANG et al., 2019)	94.27	96.92	95.57	96.44	n/a
	RequestBERT-BiLSTM	**100**	**100**	**100**	**99.74**	**99.53**
ECML/PKDD 2007	Word2Vec	63	50	56	63.46	49.87
	BERT	96	96	96	96.83	97.34
	BERT-CNN	94	94	94	94.25	97.60
	NeuralLog (LE; ZHANG, 2021)	91	90	92	93	50
	CNN (LU et al., 2018)	52	52	52	51.64	50
	RequestBERT-BiLSTM	**96**	**97**	**97**	**97.26**	**99.7**
BGL	Word2Vec	89	91	91	91.44	49.87
	BERT	100	99	99	99	99.58
	BERT-CNN	100	99	99	99.5	99.82
	NeuralLog (LE; ZHANG, 2021)	99	98	98	98	98.85
	CNN (LU et al., 2018)	93	94	95	96.37	50
	Loganomaly(MENG et al., 2019)	94	97	96	96.7	n/a
	LogBERT (GUO; YUAN; WU, 2021)	97	98	97	96.4	n/a
	RequestBERT-BiLSTM	**100**	**100**	**100**	**99.9**	**99.89**
Log-Security	Word2Vec	11	17	11	10.87	49.47
	BERT	98	97	97	97.33	97.43
	BERT-CNN	99	99	99	99.23	99.99
	NeuralLog(LE; ZHANG, 2021)	76	76	76	76.24	50
	CNN (LU et al., 2018)	26	27	26	26.80	49.87
	RequestBERT-BiLSTM	**99**	**100**	**99**	**99.51**	**99.99**

'n/a' Não há resultados no artigo original.

Neste cenário, encontram-se o *Word2Vec*, *CNN* e *NeuralLog* que a partir de uma análise baseada apenas no valor da acurácia denotam que tiveram um excelente resultados, porém o *AUC* mostra que isso é nitidamente o problema do **paradoxo de acurácia**.

A Figura 38 mostra o tempo na etapa de predição de cada modelo. Em apenas um cenário que o RequestBERT-BiLSTM não levou mais tempo que os outros. O fato do tempo ser maior para o *BGL* é devido ao seu tamanho ser muito superior aos outros conjuntos de dados testados. Já no caso do dataset Log-Security, os modelos também consumiram um tempo maior com relação aos outros datasets, tendo em vista que as requisições são de um cenário

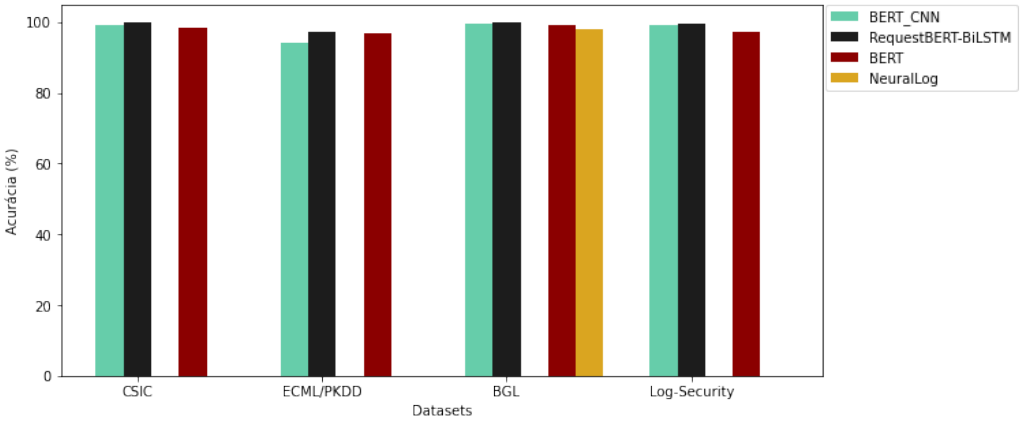
Figura 38 – Tempo de execução na predição de cada modelo. O tempo de execução de inferência foi para todo conjunto de dados de teste, em cenários reais para cada requisição o RequestBERT-BiLSTM já faria sua previsão e classificaria, com isso tornando-se mais ágil.



Fonte: Elaborado pelo autor, 2022.

real e possuem mais informações, fazendo com que o tamanho das requisições sejam maiores.

Figura 39 – Desempenhos dos modelos. Os modelos com *BERT* sempre superiores aos outros e o RequestBERT-BiLSTM sempre superior a todos. Vale ressaltar que, os modelos Word2Vec, *CNN* e NeuralLog tiveram o problema do paradoxo de acurácia.



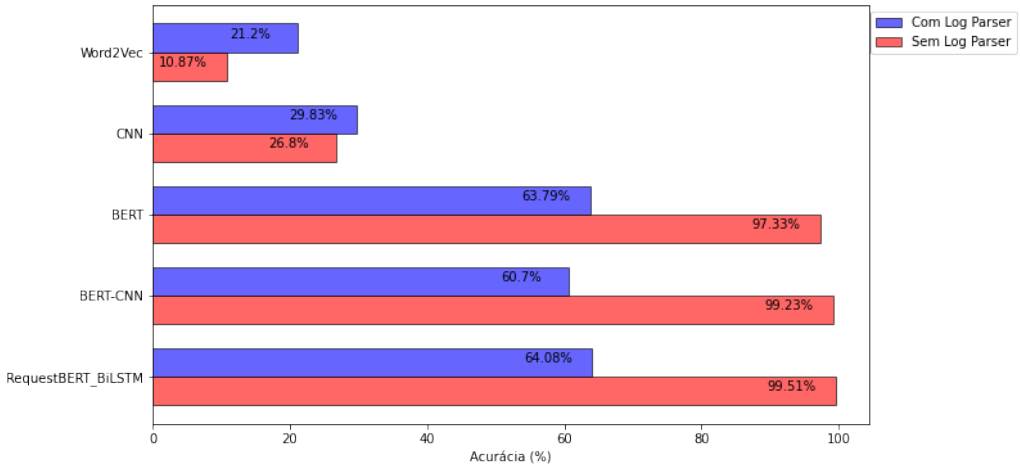
Fonte: Elaborado pelo autor, 2022.

A Figura 39 mostra a desempenho dos modelos mensurados pela Acurácia. E em todos os casos o modelo proposto neste trabalho foi superior.

A Figura 40 mostra uma compilação dos modelos testados com e sem *Log Parser*. Como resultado, a imagem deixa claro o distanciamento que houve para os modelos que utilizam o *BERT* com relação ao uso de *Log Parser* e a pequena evolução que houve para os modelos que utilizam o *word embedding*.

Essa melhora deve-se ao fato da "limpeza" que ocorre nas requisições, pois como os dois modelos *Word2Vec* e *CNN* realizam o *word embedding* das requisições com o seu *corpus* pré-treinado em uma quantidade de palavras menor, o modelo tende a diminuir o número de

Figura 40 – Comparação dos modelos com e sem *Log Parser*. A barra vermelha mostra o desempenho do modelo sem a adição do *Log Parser* e a barra azul com a adição desta etapa. Somente para os casos de *word embedding* tradicional que houve uma melhora.



Fonte: Elaborado pelo autor, 2022.

ocorrências de *Out-of-vocabulary (OOV)*, ou seja, palavras que não tem representação, logo tendo uma melhora com relação às requisições completas.

A dificuldade para os modelos com *BERT* é causada pela ausência de palavras essenciais para determinados ataques e comportamentos, e como o *BERT* utiliza o contexto isso afeta sobremaneira o desempenho do modelo.

Portanto, a execução de análise de log para o propósito deste trabalho não trouxe vantagens. A ideia que o analisador de log implementa é ideal para logs estruturados, cujos campos são fixos, como no dataset *BGL*, que apenas muda os valores conforme mostra a Figura 41. Isso é possível constatar nos trabalhos que utilizam o *BGL* e *Log Parser* como: (GUO; YUAN; WU, 2021; ZHANG et al., 2019; DU et al., 2017)

Sendo assim, para o caso dos logs que modificam a sua estrutura, como nas requisições HTTP, os métodos de analisadores de log apresentaram dificuldades para se ajustarem e compreender as requisições.

Outro ponto que vale ressaltar, é a comparação do RequestBERT-BiLSTM ao ativo *F5*

Figura 41 – Exemplo de uma entrada do Log do Dataset BGL e logo abaixo mostra os campos fixos da estrutura do Log.

```
- 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.982731
    R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error
    corrected

<Label> <Timestamp> <Date> <Node> <Time> <NodeRepeat> <Type> <Component>
    <Level> <Content>.
```

Fonte: Elaborado pelo autor, 2022.

Big-IP, no qual foi responsável por originar o conjunto de dados Log-Security baseado em seu conhecimento, e mesmo com essa prerrogativa o RequestBERT-BiLSTM conseguiu se aproximar do seu desempenho. Baseado nisso, o RequestBERT-BiLSTM possui algumas vantagens com relação ao *F5 Big-IP*: o modelo proposto é open source, adaptável a outros cenários, não é um *apliance*[2] e utiliza apenas técnicas de aprendizagem de máquina. E como desvantagem o modelo tem a incompreensão dos ataques não conhecidos ou *zero day*, o tempo de execução e a dimensionalidade de recurso computacional.

[2]Dispositivo de hardware separado e dedicado com software integrado, especificamente projetado para fornecer um recurso de computação específico.

8 CONCLUSÃO

A implementação de um sistema que realize a detecção de ataques em requisições HTTP é uma tarefa complexa. Isto ocorre tendo em vista que um ataque é criado visando enganar o servidor, misturando informações corretas com falsas, ou muitas vezes, omitindo parte dele, levando a compreensão da requisição para um cenário nebuloso entre ser verdadeira ou falsa. Além disso, somam-se a estas dificuldades a complexidade de conhecimento de técnicas de ataques, a quantidade de informações a serem analisadas e a importância do tratamento das requisições se devem ou não haver a análise automatizada.

Este trabalho apresentou o RequestBERT-BiLSTM, um modelo de detecção de ataques em requisições HTTP baseado em *BERT* e *BiLSTM*, que dado um conjunto de requisições HTTP é possível identificar ataques.

Na etapa do pré-processamento, o RequestBERT-BiLSTM considera toda a requisição HTTP como entrada sem a necessidade de *Log Parsing* eliminando os caracteres especiais, sendo oposto a trabalhos que apenas tratam partes consideradas mais importantes das requisições.

Em seguida, buscou-se executar a análise das requisições HTTP e mostrar a dificuldade dessa tarefa para a detecção de ataques. A partir disto, foram realizados experimentos com os analisadores de log em diferentes abordagens para expor a dificuldade que esta atividade pode trazer ao cenário de requisições HTTP. A realização da etapa de análise de log se mostrou interessante para elucidar a dificuldade que a automatização pode trazer para um cenário onde os dados se modificam constantemente.

A metodologia proposta foi aplicada em quatro conjuntos de dados, onde um dos conjuntos de dados corresponde a dados reais de acessos às páginas hospedadas em uma instituição federal. E baseado nos experimentos realizados, o RequestBERT-BiLSTM se mostrou eficiente em todos os cenários de classificação em comparação aos modelos experimentados e de trabalhos do estado da arte. Outra evidência da complexidade na detecção de ataques em requisições HTTP pode ser observada nos resultados analisando o desempenho individual de classificadores que, muitas vezes, têm os seus resultados próximos do que se espera de uma classificação aleatória.

De maneira geral, conclui-se que os principais fatores que levaram o RequestBERT-BiLSTM a ser superior aos modelos desenvolvidos neste trabalho e as do estado da arte foram: não exclusão de dados relevantes, a não utilização do *Log Parser* e a utilização do *BERT* com

BiLSTM.

8.1 CONTRIBUIÇÕES

Com base nos resultados apresentados, a contribuição desse trabalho se destaca no problema de detecção de ataques em requisições HTTP e detecção de anomalias em log. O modelo proposto nesta dissertação atingiu um valor de Acurácia de 99.9% em uma classificação binária e 99.51% em uma classificação com múltiplas classes e sempre ficando com AUC acima de 97%.

Vale citar, que os resultados apresentados neste trabalho superaram as trabalhos já publicados. Outra conclusão baseada nos experimentos é que o RequestBERT-BiLSTM trouxe mais uma contribuição para a área de aplicação ao utilizar o *BERT*, no qual em nenhum trabalho relacionado utilizou.

O principal reconhecimento do avanço que este trabalho alcançou e de seu pioneirismo com a utilização do *BERT* e *BiLSTM* em requisições HTTP foi a aceitação do artigo *LogBERT-BiLSTM: Detecting Malicious Web Requests* na *International Conference on Artificial Neural Networks (ICANN)* e o artigo *Detecting malicious HTTP requests without Log Parser using RequestBERT-BiLSTM* no *Brazilian Conference on Intelligent Systems (BRACIS)* ambos em 2022, isto demostra a importância deste trabalho tanto para a literatura quanto para cenários reais.

A diferença principal em ambos é que o *Detecting malicious HTTP requests without Log Parser using RequestBERT-BiLSTM* demonstra a dificuldade que o *Log Parser* produz para requisições HTTP e apresenta a eficiência do modelo baseado na junção do *BERT* com *BiLSTM* para requisições HTTP e outros tipos de log. Já o *LogBERT-BiLSTM: Detecting Malicious Web Requests* não trata sobre *Log Parser* e sim na inovação do *BERT* com *BiLSTM* para o cenário de requisição HTTP.

8.2 TRABALHOS FUTUROS

Como trabalhos futuros e extensão dessa pesquisa, destacam-se a investigação e exploração em diferentes conjuntos de dados e utilização de técnicas de aprendizado de máquina não supervisionado, dado que isso não foi abordado.

O RequestBERT-BiLSTM também pode ser aplicado, salvo ajustes, para ao invés de

considerar somente a requisição, considerar também a resposta do servidor Web, tentando criar uma associação desta com uma possível tentativa de ataque de dia zero. Em que seria possível a detecção de varreduras iniciais consideradas requisições legítimas, visando encontrar vulnerabilidades em ferramentas e tecnologias instaladas nos servidores Web, podendo isso ser descoberto com adição das respostas dos servidores no aprendizado do modelo. Baseado nisso, pode-se utilizar algumas características iniciais para um possível ataque de dia zero: o número de solicitações em um curto espaço de tempo, a localização geográfica do IP, requisições que buscam funcionalidades específicas de softwares, nomes de ferramentas tradicionais de scanner de vulnerabilidade, entre outros.

REFERÊNCIAS

ALTHUBITI, S.; YUAN, X.; ESTERLINE, A. Analyzing http requests for web intrusion detection. 10 2017.

BACH-NUTMAN, M. Understanding the top 10 OWASP vulnerabilities. *CoRR*, abs/2012.09960, 2020. Disponível em: <https://arxiv.org/abs/2012.09960>.

BOJANOWSKI, P.; GRAVE, E.; JOULIN, A.; MIKOLOV, T. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. Disponível em: <http://arxiv.org/abs/1607.04606>.

CASPIO. *Parameters as Query String Values*. 2022. Online; acessado 21 Março 2022. Disponível em: <https://howto.caspio.com/parameters/parameters-as-query-string-values/>.

CECCON, D. *BERT, o modelo de processamento de linguagem natural que revolucionou a área*. 2020. Online; acessado 14 Junho 2022. Disponível em: <https://iaexpert.academy/2020/04/27/bert-o-modelo-de-processamento-de-linguagem-natural-que-revolucionou-a-area/>.

CHALAPATHY, R.; CHAWLA, S. Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407, 2019. Disponível em: <http://arxiv.org/abs/1901.03407>.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM Comput. Surv.*, v. 41, 07 2009.

CHEN, Z.; LIU, J.; GU, W.; SU, Y.; LYU, M. R. Experience report: Deep learning-based system log analysis for anomaly detection. *CoRR*, abs/2107.05908, 2021. Disponível em: <https://arxiv.org/abs/2107.05908>.

CHOLLET, F. *Deep Learning with Python*. Manning Publications Company, 2017. ISBN 9781617294433. Disponível em: <https://books.google.com.br/books?id=Yo3CAQAACAAJ>.

CISCO. *What Is a Cyberattack?* 2022. [Online; acessado 7-Marc-2022]. Disponível em: <https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html>.

DENG, L.; LIU, Y. *Deep Learning in Natural Language Processing*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2018. ISBN 9789811052088.

DEVELOPERS, G. *Embeddings: Translating to a Lower-Dimensional Space*. 2022. Online; acessado 02 Dezembro 2022. Disponível em: <https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space>.

DEVLIN, J.; CHANG, M.; LEE, K.; TOUTANOVA, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. Disponível em: <http://arxiv.org/abs/1810.04805>.

DU, M.; LI, F. Spell: Streaming parsing of system event logs. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. [S.l.: s.n.], 2016. p. 859–864.

DU, M.; LI, F.; ZHENG, G.; SRIKUMAR, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: . [s.n.], 2017. p. 1285–1298. ISBN 9781450349468. Disponível em: <https://doi.org/10.1145/3133956.3134015>.

F5. *Secure against the OWASP Top 10 for 2021*. 2022. Online; acessado 01 Dezembro 2022. Disponível em: <https://support.f5.com/csp/article/K45215395>.

F5-ATTACK. *Assigning Attack Signatures to Security Policies*. 2022. [Online; acessado 20-Fev-2022]. Disponível em: <https://techdocs.f5.com/kb/en-us/products/big-ip_asm/manuals/product/asm-bot-and-attack-signatures-13-0-0/1.html>.

F5BIG-IP. *What Is the BIG-IP System?* 2022. [Online; acessado 22-Fev-2022]. Disponível em: <https://techdocs.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/tmos-concepts-11-5-0/1.html>.

FORTIGATE. *Global Threat Landscape Report*. [S.l.], 2022. Disponível em: <https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/report-q1-2022-threat-landscape.pdf>.

FORTINET. *Types of Cyber Attacks*. 2022. [Online; acessado 12-Marc-2022]. Disponível em: <https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks>.

GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. <http://www.deeplearningbook.org>.

GRAVES, A.; JAITLY, N.; MOHAMED, A.-r. Hybrid speech recognition with deep bidirectional lstm. In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. [S.l.: s.n.], 2013. p. 273–278.

GUO, H.; YUAN, S.; WU, X. Logbert: Log anomaly detection via bert. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2021. p. 1–8.

HE, P.; ZHU, J.; HE, S.; LI, J.; LYU, M. R. An evaluation study on log parsing and its use in log mining. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. [S.l.: s.n.], 2016. p. 654–661.

HE, P.; ZHU, J.; HE, S.; LI, J.; LYU, M. R. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 15, n. 06, p. 931–944, nov 2018. ISSN 1941-0018.

HE, P.; ZHU, J.; ZHENG, Z.; LYU, M. R. Drain: An online log parsing approach with fixed depth tree. In: *2017 IEEE International Conference on Web Services (ICWS)*. [S.l.: s.n.], 2017. p. 33–40.

HE, S.; ZHU, J.; HE, P.; LYU, M. R. Experience report: System log analysis for anomaly detection. In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.: s.n.], 2016. p. 207–218.

HE, S.; ZHU, J.; HE, P.; LYU, M. R. Loghub: A large collection of system log datasets towards automated log analytics. *CoRR*, abs/2008.06448, 2020. Disponível em: <https://arxiv.org/abs/2008.06448>.

ITO, M.; IYATOMI, H. Web application firewall using character-level convolutional neural network. In: *2018 IEEE 14th International Colloquium on Signal Processing Its Applications (CSPA)*. [S.l.: s.n.], 2018. p. 103–106.

JACKSON, P.; MOULINIER, I. *Natural language processing for online applications: text retrieval, extraction and categorization*. [S.l.]: John Benjamins Philadelphia, 2007.

JIANG, Z.; HASSAN, A. E.; FLORA, P.; HAMANN, G. Abstracting execution logs to execution events for enterprise applications (short paper). In: . [S.l.: s.n.], 2008. p. 181 – 186. ISBN 978-0-7695-3312-4.

JIANG, Z.; HASSAN, A. E.; HAMANN, G.; FLORA, P. An automated approach for abstracting execution logs to execution events. *Journal of Software Maintenance*, v. 20, p. 249–267, 07 2008.

KARAMITSOS, I.; AFZULPURKAR, A.; TRAFALIS, T. Malware detection for forensic memory using deep recurrent neural networks. *Journal of Information Security*, v. 11, p. 103–120, 01 2020.

KIM, J.; KIM, J.; THU, H. L. T.; KIM, H. Long short term memory recurrent neural network classifier for intrusion detection. In: *2016 International Conference on Platform Technology and Service (PlatCon)*. [S.l.: s.n.], 2016. p. 1–5.

KIM, Y. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. Disponível em: <http://arxiv.org/abs/1408.5882>.

KUANG, X.; ZHANG, M.; LI, H.; ZHAO, G.; CAO, H.; WU, Z.; WANG, X. Deepwaf: Detecting web attacks based on cnn and lstm models. In: *Cyberspace Safety and Security: 11th International Symposium, CSS 2019, Guangzhou, China, December 1–3, 2019, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2019. p. 121–136. ISBN 978-3-030-37351-1. Disponível em: <https://doi.org/10.1007/978-3-030-37352-8_11>.

LE, V.; ZHANG, H. Log-based anomaly detection without log parsing. *CoRR*, abs/2108.01955, 2021. Disponível em: <https://arxiv.org/abs/2108.01955>.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LIU, J.; SONG, X.; ZHOU, Y.; PENG, X.; ZHANG, Y.; LIU, P.; WU, D.; ZHU, C. Deep anomaly detection in packet payload. *Neurocomputing*, v. 485, p. 205–218, 2022. ISSN 0925-2312. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0925231221016349>.

LU, S.; WEI, X.; LI, Y.; WANG, L. Detecting anomaly in big data system logs using convolutional neural network. In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. [S.l.: s.n.], 2018. p. 151–158.

MDN. *Mensagens HTTP*. 2021. Online; acessado 10 Março 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Messages>.

MENG, W.; LIU, Y.; ZHU, Y.; ZHANG, S.; PEI, D.; LIU, Y.; CHEN, Y.; ZHANG, R.; TAO, S.; SUN, P.; ZHOU, R. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: *IJCAI*. [S.l.: s.n.], 2019.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, v. 2013, 01 2013.

MITRE. *Enterprise Matrix*. 2022.

NAGAPPAN, M.; VOUK, M. A. Abstracting log lines to log event types for mining software system logs. In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. [S.l.: s.n.], 2010. p. 114–117.

NEDELKOSKI, S.; BOGATINOVSKI, J.; ACKER, A.; CARDOSO, J.; KAO, O. Self-supervised log parsing. In: _____. [S.l.: s.n.], 2021. p. 122–138. ISBN 978-3-030-67666-7.

ODUMUYIWA, V.; CHIBUEZE, A. Automatic detection of http injection attacks using convolutional neural network and deep neural network. *J. Cyber Secur. Mobil.*, v. 9, p. 489–514, 2020.

OINKINA, H. *Understanding LSTM networks*. 2015. [Online; acessado 18-Jan-2022]. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

OLINER, A.; STEARLEY, J. What supercomputers say: A study of five system logs. In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. [S.l.: s.n.], 2007. p. 575–584.

OWASP. *OWASP Top Ten*. 2020. Online; acessado 01 Dezembro 2022. Disponível em: <https://owasp.org/www-project-top-ten/>.

PANG, G.; SHEN, C.; CAO, L.; HENGEL, A. V. D. Deep learning for anomaly detection. *ACM Computing Surveys*, Association for Computing Machinery (ACM), v. 54, n. 2, p. 1–38, mar 2022. Disponível em: <https://doi.org/10.1145%2F3439950>.

POSTON, H. *Top threat modeling frameworks: STRIDE, OWASP Top 10, MITRE ATTCK framework and more*. 2021.

RAÏSSI, C.; BRISSAUD, J.; DRAY, G.; PONCELET, P.; ROCHE, M.; TEISSEIRE, M. Web analyzing traffic challenge: Description and results. In: . [S.l.: s.n.], 2007.

RONG, W.; ZHANG, B.; LV, X. Malicious web request detection using character-level CNN. *CoRR*, abs/1811.08641, 2018. Disponível em: <http://arxiv.org/abs/1811.08641>.

TALON. *Deep Learning Book*. 2022. [Online; acessado 22-Fev-2022]. Disponível em: <https://www.deeplearningbook.com.br/o-que-sao-redes-neurais-artificiais-profundas/>.

TANG, L.; LI, T.; PERNG, C.-S. Logsig: Generating system events from raw textual logs. In: . New York, NY, USA: Association for Computing Machinery, 2011. ISBN 9781450307178. Disponível em: <https://doi.org/10.1145/2063576.2063690>.

TORRANO-GIMENEZ, C.; PEREZ-VILLEGAS, A.; ALVAREZ, G. A self-learning anomaly-based web application firewall. In: . [S.l.: s.n.], 2009. v. 63, p. 85–92. ISBN 978-3-642-04090-0.

UDDIN, M. F. Addressing accuracy paradox using enhanched weighted performance metric in machine learning. In: *2019 Sixth HCT Information Technology Trends (ITT)*. [S.l.: s.n.], 2019. p. 319–324.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. Disponível em: <http://arxiv.org/abs/1706.03762>.

XUAN, C.; DINH, H.; VICTOR, T. Malicious url detection based on machine learning. *International Journal of Advanced Computer Science and Applications*, v. 11, 01 2020.

YU, L.; CHEN, L.; DONG, J.; LI, M.; LIU, L.; ZHAO, B.; ZHANG, C. Detecting malicious web requests using an enhanced textcnn. In: *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. [S.l.: s.n.], 2020. p. 768–777.

ZHANG, X.; XU, Y.; LIN, Q.; QIAO, B.; ZHANG, H.; DANG, Y.; XIE, C.; YANG, X.; CHENG, Q.; LI, Z.; CHEN, J.; HE, X.; YAO, R.; LOU, J.-G.; CHINTALAPATI, M.; SHEN, F.; ZHANG, D. Robust log-based anomaly detection on unstable log data. In: . [s.n.], 2019. p. 807–817. ISBN 9781450355728. Disponível em: <https://doi.org/10.1145/3338906.3338931>.

ZHU, J.; HE, S.; LIU, J.; HE, P.; XIE, Q.; ZHENG, Z.; LYU, M. R. Tools and benchmarks for automated log parsing. *CoRR*, abs/1811.03509, 2018. Disponível em: <http://arxiv.org/abs/1811.03509>.

ZHU, J.; HE, S.; LIU, J.; HE, P.; XIE, Q.; ZHENG, Z.; LYU, M. R. Tools and benchmarks for automated log parsing. In: . IEEE Press, 2019. Disponível em: <https://doi.org/10.1109/ICSE-SEIP.2019.00021>.

ZOLOTUKHIN, M.; HäMäLäINEN, T.; KOKKONEN, T.; SILTANEN, J. Analysis of http requests for anomaly detection of web attacks. In: *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. [S.l.: s.n.], 2014. p. 406–411.