

UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO APLICADA
PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Richard Caio Silva Rego

**APLICAÇÃO COMBINADA DO FILTRO DE BLOOM COM REDES
NEURAS RECORRENTES PARA DETECÇÃO DE ATAQUES WEB**

Santa Maria
2020

Richard Caio Silva Rego

**APLICAÇÃO COMBINADA DO FILTRO DE BLOOM COM REDES NEURAIIS
RECORRENTES PARA DETECÇÃO DE ATAQUES WEB**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Maria como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Raul Ceretta Nunes

Santa Maria
2020

Rego, Richard Caio Silva
Aplicação combinada do Filtro de Bloom com Redes
Neurais Recorrentes para detecção de ataques web /
Richard Caio Silva Rego.- 2020.
69 p.; 30 cm

Orientador: Raul Ceretta Nunes
Dissertação (mestrado) - Universidade Federal de Santa
Maria, Centro de Tecnologia, Programa de Pós-Graduação em
Ciência da Computação , RS, 2020

1. Ataques Web 2. Filtro de Bloom 3. Redes Neurais
Recorrentes I. Nunes, Raul Ceretta II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.


Declaro, RICHARD CAIO SILVA REGO, para os devidos fins e sob as penas da lei, que a pesquisa constante neste trabalho de conclusão de curso (Dissertação) foi por mim elaborada e que as informações necessárias objeto de consulta em literatura e outras fontes estão devidamente referenciadas. Declaro, ainda, que este trabalho ou parte dele não foi apresentado anteriormente para obtenção de qualquer outro grau acadêmico, estando ciente de que a inveracidade da presente declaração poderá resultar na anulação da titulação pela Universidade, entre outras consequências legais.

Richard Caio Silva Rego

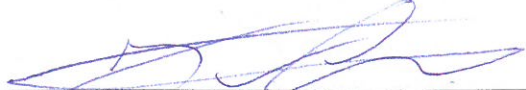
**Aplicação combinada do filtro de Bloom com Redes Neurais Recorrentes para
detecção de ataques web**

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PGCC) da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

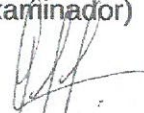
Aprovado em 27 de Janeiro de 2020:



Raul Ceretta Nunes, Dr.
(Presidente/Orientador)



Joaquim Vinícius Carvalho Assunção, Dr.
(Examinador)



Cristian Ramón Cappelletti, Dr. (videoconferência)
(Examinador)

Santa Maria
2020

*Dedico à minha esposa e fiel companheira
Heliene, aos meus pais Risonaldo e Clarisse
e especialmente à minha avó Nezita.*

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus pela dádiva da vida. Agradeço à minha esposa Heliene, fiel companheira e minha maior incentivadora, por me acompanhar de perto nessa longa jornada. Todo o meu amor a você.

A toda minha família pelas orações e palavras de apoio que sempre recebi vindos dos mais diversos cantos do Brasil. Em especial, à minha tia Jucélia, minha primeira educadora, que conduziu meus primeiros passos pelo caminho do conhecimento. Ao meu pai, Risonaldo, homem de grande coração, honesto e trabalhador. E minha mãe, Clarisse, mulher guerreira e muito corajosa.

Ao meu orientador, Professor Dr. Raul Ceretta Nunes, pela excelente condução deste trabalho de pesquisa com serenidade e profissionalismo. Agradeço por sua compreensão nos momentos de dificuldades e pelas palavras de incentivo. Agradeço também aos professores, Dr. Cristian Cappo e Dr. Joaquim Assunção pelo apoio e orientações que me auxiliaram na condução final deste trabalho.

Meus agradecimentos aos professores, alunos e técnicos da UFSM que fizeram parte da minha jornada nesta prestigiada instituição. Aos colegas do Grupo de Pesquisa *GTSeg* com os quais convivi pelos últimos dois anos.

Aos amigos que torceram por mim nesse período de grandes desafios e todos aqueles que de alguma forma contribuíram para que eu pudesse alcançar esta vitória.

Muito obrigado.

RESUMO

APLICAÇÃO COMBINADA DO FILTRO DE BLOOM COM REDES NEURAIIS RECORRENTES PARA DETECÇÃO DE ATAQUES WEB

AUTOR: RICHARD CAIO SILVA REGO
ORIENTADOR: RAUL CERETTA NUNES

A diversidade e a complexidade de ataques que ameaçam as aplicações Web têm aumentado nos últimos anos ocasionando prejuízos aos usuários e proprietários desses aplicativos. Para incrementar os níveis de segurança, especialistas têm adotado o uso de sistemas de detecção de intrusão que atuam na camada de aplicação, especialmente aqueles baseados em anomalias. As técnicas de detecção de anomalias visam estender os sistemas de segurança agregando meios para a detecção de ataques ainda desconhecidos que não seriam descobertos por sistemas de detecção baseados em assinatura. Técnicas de aprendizado de máquina são largamente utilizadas em detectores, particularmente as redes neurais recorrentes, que vêm se destacando nos últimos anos por seu bom desempenho na tarefa de detecção de ataques web. No entanto, as pesquisas com redes recorrentes têm focado no aumento do desempenho preditivo dos modelos. Além disso, as técnicas baseadas em aprendizado profundo apresentam elevado custo computacional. Portanto, faz-se necessário projetar modelos de detecção de intrusão que sejam eficazes do ponto de vista preditivo, mas também que sejam eficientes quanto ao tempo de detecção. Neste trabalho foi utilizado o Filtro de Bloom como ferramenta de apoio ao detector baseado em rede neural recorrente. O filtro de Bloom é uma ferramenta mais ágil e atua na seleção de instâncias reduzindo os dados enviados à rede neural. Esta combinação proporcionou uma redução no tempo médio de detecção sem afetar as métricas de detecção das redes recorrentes. Além disso, com a avaliação comparativa entre as redes recorrentes dos tipos LSTM, BI-LSTM e GRU foi possível avaliar a melhor variante para o modelo. A combinação entre Filtro de Bloom e Rede Neural Recorrente mostrou-se adequada e eficaz para a detecção de ataques contra aplicações Web apresentando um bom desempenho preditivo e tempo médio de detecção se comparado aos cenários sem a presença do filtro.

Palavras-chave: Ataques Web. Filtro de Bloom. Redes Neurais Recorrentes.

ABSTRACT

Bloom filter application with recurrent neural networks for web attack detection

AUTHOR: RICHARD CAIO SILVA REGO

ADVISOR: RAUL CERETTA NUNES

The diversity and complexity of attacks that threaten Web applications have grown in recent years, causing losses for users and owners of these applications. In order to increase security levels, experts have implemented intrusion detection systems to act on the application layer, especially those based on anomalies. Anomaly detection techniques aim to extend security systems by adding alternatives for detecting unknown attacks that would not be discovered by signature-based detection systems. Machine learning methods are widely used in detectors, mainly recurrent neural networks, which have gained great importance in recent years due to their great performance in the task of detecting web attacks. However, research with recurrent networks has focused on increasing the prediction performance of the models. In addition, techniques based on deep learning have shown high computational cost. Therefore, it is necessary to design effective intrusion detection models for the forecasting task, as well as efficient in terms of detection time. In this work, the Bloom Filter was used as a tool to support the detector based on recurrent neural network. The Bloom filter is a more agile tool and acts in the selection of instances, reducing the input data of the neural network. This combination provided a reduction in the average detection time without affecting the detection metrics results of recurring networks. In addition, with the comparative assessment between recurrent networks of the LSTM, BI-LSTM and GRU types, it was possible to evaluate the best variant for the model. The combination of Bloom Filter and Recurrent Neural Network proved to be adequate and effective for detecting attacks against Web applications, presenting a great predictive performance and average detection time when compared to scenarios without the presence of the filter.

Keywords: Web attacks. Bloom filter. Recurrent Neural Networks.

LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de requisição HTTP com injeção de código malicioso na consulta SQL.	23
Figura 2 – Exemplo de inserção de caracteres aleatórios em ataque de Estouro de Buffer.	23
Figura 3 – Exemplo de um ataque XSS com a inerção de um script malicioso. .	24
Figura 4 – Exemplo de um script que pode ser inserido em requisição para ataque de Injeção SSI	24
Figura 5 – Célula <i>Long Short-Term Memory</i> - LSTM	26
Figura 6 – Célula <i>Gated Recurrent Unit</i> - GRU	27
Figura 7 – Célula LSTM Bidirecional - BI-LSTM	28
Figura 8 – Descida do gradiente com a) uma taxa de aprendizagem pequena e b) taxa de aprendizagem grande	29
Figura 9 – (a) Filtro vazio (b) Preenchimento do filtro (c) Consulta de elementos no filtro.	31
Figura 10 – Fluxo da informação no modelo de detecção	33
Figura 11 – Exemplos de requisições dos tipos POST e GET	34
Figura 12 – Exemplo de extração de recursos numéricos de uma requisição HTTP	36
Figura 13 – Redução do valor de P aumenta o tamanho do filtro e a quantidade de funções hashes.	38
Figura 14 – Matriz de confusão	43
Figura 15 – Quantidade de colisões no filtro de acordo com a configuração . . .	45
Figura 16 – Filtro de Bloom expandido com tamanho dobrado em relação à configuração padrão	46
Figura 17 – Gráfico com resultados da métrica de Acurácia (ACC)	47
Figura 18 – Gráfico com resultados da métrica de Precisão (P)	48
Figura 19 – Gráfico com resultados da métrica de Taxa de Detecção (DR)	49
Figura 20 – Gráfico com resultados da métrica de Tempo Médio de Detecção (AVG)	50
Figura 21 – Comparativo de tempo de detecção entre as redes recorrentes em cenários com e sem o Filtro de Bloom	56

LISTA DE TABELAS

Tabela 1 – Trabalhos relacionados	19
Tabela 2 – Exemplo de uma requisição e os valores extraídos a partir da seleção de atributos	35
Tabela 3 – Métricas de desempenho preditivo	43
Tabela 4 – Configurações dos Filtros Bloom padrão e expandido	46
Tabela 5 – Resultados do GridSearchCV para hiperparâmetros das redes neurais.	54
Tabela 6 – Comparativo entre índices de predição para as redes neurais recorrentes	55
Tabela 7 – Ranking da melhor rede baseado na métrica ARR	57
Tabela 8 – Tipos e quantidades de dados normais e ataques presentes no conjunto CSIC 2012.	58
Tabela 9 – Resultados de detecção por tipo de ataque aplicado às redes neurais recorrentes	59

SUMÁRIO

1	INTRODUÇÃO	12
1.1	ORGANIZAÇÃO DO TRABALHO	13
2	TRABALHOS RELACIONADOS	15
2.1	TRABALHOS COM APRENDIZADO DE MÁQUINA TRADICIONAL E COMPARATIVOS	15
2.2	TRABALHOS COM REDES RECORRENTES	16
2.3	TRABALHOS COM EMPILHAMENTO DE TÉCNICAS	18
2.4	RESUMO DOS TRABALHOS	18
3	FUNDAMENTAÇÃO TEÓRICA	22
3.1	ATAQUES CONTRA APLICAÇÕES WEB	22
3.2	REDES NEURAIS RECORRENTES	25
3.2.1	Long Short Term Memory.	25
3.2.2	Gated Recurrent Unit	26
3.2.3	LSTM Bidirecional	27
3.2.4	Configuração de Redes Neurais Recorrentes	28
3.3	FILTRO DE BLOOM	30
4	MODELO FILTRO DE BLOOM COM REDE NEURAL RECORRENTE (BLOOM-RNN)	32
4.1	FLUXO DE INFORMAÇÕES NO MODELO DE DETECÇÃO	32
4.2	PRÉ-PROCESSAMENTO DOS DADOS	33
4.2.1	Limpeza dos dados	34
4.2.2	Seleção de Recursos	34
4.3	ESTÁGIO 1: FILTRO DE BLOOM.	36
4.3.1	Dinâmica do Filtro	37
4.3.2	Eficiência do Filtro	38
4.4	ESTÁGIO 2: REDE NEURAL RECORRENTE	39
4.4.1	Treinamento da rede neural	39
4.4.2	Classificação	40
4.5	CONSIDERAÇÕES DO CAPÍTULO	40
5	FILTRO DE BLOOM COMO FERRAMENTA DE APOIO AO DETECTOR	41
5.1	PLANEJAMENTO DOS EXPERIMENTOS	41
5.1.1	Conjunto de Dados CSIC 2010	42
5.1.2	Métricas	42

5.2	IMPLEMENTAÇÃO E CONFIGURAÇÃO DOS DETECTORES	43
5.2.1	Detecção com Filtro de Bloom (D-BLOOM)	44
5.2.2	Detecção com Filtro de Bloom Expandido (D-BLOOM+)	44
5.2.2.1	<i>Otimização das configurações do Filtro Bloom</i>	45
5.2.2.2	<i>Detecção com o Filtro Expandido</i>	46
5.3	RESULTADOS E DISCUSSÃO	47
5.4	CONSIDERAÇÕES DO CAPÍTULO	50
6	ANÁLISE DE REDES RECORRENTES APLICADAS AO MODELO DE DETECÇÃO	52
6.1	ORGANIZAÇÃO DOS EXPERIMENTOS	53
6.1.1	Seleção de hiperparâmetros para as redes neurais recorrentes .	53
6.2	ANÁLISE DAS REDES RECORRENTES NO MODELO BLOOM-RNN	54
6.3	ANÁLISE DE DESEMPENHO DO MODELO POR ATAQUE	57
6.4	CONSIDERAÇÕES DO CAPÍTULO	60
7	CONCLUSÕES E TRABALHOS FUTUROS	62
7.1	CONCLUSÕES	62
7.2	TRABALHOS FUTUROS	63
	Referências	65

1 INTRODUÇÃO

Os ataques maliciosos contra aplicações Web estão se tornando mais complexos e diversificados. Estes ataques podem causar sérios danos ao funcionamento desses serviços, acarretando prejuízos sociais e econômicos aos proprietários e usuários (AHMED; MAHMOOD; HU, 2016). A diversidade de ataques voltados para aplicações Web é grande. Entre os mais comuns nesse domínio pode-se citar ataques de *SQL Injection* (SQLi), *Content Spoofing*, *Cross-Site Scripting* (XSS), *Buffer Overflow*, falsificação de solicitação entre sítios (web), *Denial Of Service*, *OS Command Injection*, *Path Transversal* ou *LDAP injection* (LDAPi) (GIMÉNEZ et al., 2015).

Para incrementar os níveis de segurança nas aplicações, especialistas recomendam a utilização de sistemas de detecção de intrusões (SDIs), especialmente aqueles baseados na detecção de anomalias que suportam a detecção de novos ataques (KRUEGEL; VIGNA, 2003). As técnicas de detecção de anomalias visam estender os sistemas de segurança agregando meios para a detecção de ataques desconhecidos. Na literatura, vários modelos de detecção de intrusão específicos para aplicações Web são desenvolvidos com base no aprendizado de máquina (ALTHUBITI; YUAN; ESTERLINE, 2017)(SMITHA; HAREESHA; KUNDAPUR, 2019a)(ITO; IYATOMI, 2018)(EPP; FUNK; CAPPO, 2017).

Redes Neurais Recorrentes (*Recurrent Neural Networks* - RNNs) são uma categoria de rede neural artificial, inseridas no campo do aprendizado profundo, projetadas para reconhecer padrões em sequências de dados, como texto, dados de sensores, ou informações em série que estabeleçam um ordenamento lógico (LE; KIM; KIM, 2017). As RNNs são propícias para trabalhar com grande volume de dados e vêm apresentando bons resultados no domínio de detecção de ataques Web. Recentemente, alguns trabalhos de pesquisa começaram a explorar o uso de diferentes categorias de redes recorrentes em modelos de detecção de intrusão com bom desempenho como em (BOCHEM; ZHANG; HOGREFE, 2017) que explora redes *Long Short-Term Memory* (LSTM), (HAO; LONG; YANG, 2019) que utiliza redes *Bidirectional Long Short-Term Memory* (BI-LSTM) e (LIANG; ZHAO; YE, 2017a) que se apropria dos recursos das redes *Gated Recurrent Unit* (GRU).

No entanto, estas pesquisas têm focado especificamente no aumento do desempenho preditivo dos modelos, impulsionadas pelos constantes avanços de poder computacional. Algumas propostas vão além na busca por índices maiores de acurácia propondo técnicas que combinam duas ou mais redes recorrentes (LIANG; ZHAO; YE, 2017b) ou que combinam variadas categorias de redes neurais em um mesmo modelo como em (KIM; CHO, 2018) e (ITO; IYATOMI, 2018). Para modelos baseados em aprendizado de máquina um pequeno incremento na precisão afeta consideravelmente o tempo computacional (ONEY; PEKER, 2018).

Para Giménez et al. (2015) sistemas de detecção de intrusão voltados para

aplicações da Web devem ser projetados para serem eficazes e eficientes ao mesmo tempo. Sua eficácia está relacionada à capacidade de detectar o maior número de ataques com um baixo número de falso alerta. Já para ser um IDS eficiente, o sistema precisa ter baixa complexidade computacional e baixo consumo de recursos. Portanto, faz-se necessário propor soluções que possam equilibrar eficiência e eficácia.

Neste sentido, uma estratégia utilizada na construção de detectores de intrusões é a redução da dimensionalidade dos dados como abordado em (HERRERA-SEMENETS et al., 2018) e (ZHOU et al., 2014). O filtro de Bloom (BLOOM, 1970) tem-se mostrado como excelente ferramenta de apoio a detectores permitindo reduzir o tempo de detecção de ataques de redes, inclusive, com redes neurais recorrentes (ZHOU et al., 2014; FENG; LI; CHANA, 2017).

Este trabalho propõe o BLOOM-RNN, um método para detecção de ataques em aplicações Web que combina o Filtro de Bloom com redes neurais recorrentes. O método explora a aplicação do Filtro de Bloom como ferramenta de apoio ao detector reduzindo a quantidade de requisições enviadas ao detector. Também é proposta a experimentação de três diferentes redes neurais recorrentes (LSTM, BI-LSTM e GRU), a busca pela melhor configuração (hiperparâmetros) para cada uma delas e indicação da mais adequada ao modelo.

A exploração da combinação do Filtro de Bloom com Redes Recorrentes é a principal contribuição deste trabalho, diante da necessidade de se implementar modelos de acuracidade eficaz e eficiência computacional. Poucos trabalhos se atêm às medidas que considerem o tempo de detecção e o seu impacto nos servidores da Web quando apresentam seus modelos de detecção de ataques. Entende-se contribuir efetivamente para o campo de detecção de anomalias com um modelo que consegue reduzir o tempo de detecção sem afetar o desempenho preditivo da rede neural recorrente.

Outra importante contribuição está no comparativo entre o desempenho das diferentes redes neurais recorrentes para detecção de ataques que afligem às aplicações Web. A comparação de entre três categorias de redes neurais recorrentes permite avaliar qual é mais indicada para o modelo. Essa análise é feita sob aspecto da predição, do tempo de detecção e a melhor delas se consideradas essas duas medidas ao mesmo tempo. A realização de testes individuais por tipo de ataque complementa esta análise das redes recorrentes.

1.1 ORGANIZAÇÃO DO TRABALHO

O texto deste trabalho está organizado da seguinte forma: No Capítulo 2 são apresentados os trabalhos atuais que contribuíram para o conhecimento relacionado e o desenvolvimento dos experimentos propostos nesta dissertação. Os trabalhos relacionados estão organizados em seções com artigos que envolvem a utilização de aprendizado de máquina tradicional e estudos comparativos 2.1, trabalhos com redes

neurais recorrentes 2.2 e métodos que assimilaram mais de uma técnica de machine learning em seus modelos 2.3.

O Capítulo 3 apresenta conceitos fundamentais sobre ataques contra aplicações da Web. Trata também das duas principais ferramentas utilizadas neste trabalho. A Seção 3.2 destaca os fundamentos sobre as redes neurais recorrentes e algumas de suas vertentes. A Seção 3.3 apresenta os fundamentos acerca do Filtro de Bloom.

O Capítulo 4 expõe um detalhamento do modelo proposto por este trabalho. Primeiramente, as Seções 4.1 e 4.2 apresentam como se dá o fluxo de informações dentro do modelo e o pré-processamento dos dados, respectivamente. Em seguida, são discutidos os estágios do modelo que contemplam o uso do Filtro de Bloom 4.3 e a Rede Neural Recorrente 4.4.

O Capítulo 5 apresenta os experimentos realizados para comprovar a eficiência do Filtro de Bloom quando aplicado como ferramenta de filtragem diante de técnicas de classificação baseadas em aprendizado de máquina. O Capítulo 6 faz uma análise minuciosa de três modelos de redes neurais recorrentes aplicadas ao segundo estágio do modelo proposto neste trabalho.

Por fim, o Capítulo 7 apresenta as conclusões derivadas deste trabalho de pesquisa e aponta possíveis trabalhos que possam avançar nas pesquisas realizadas por esta dissertação.

2 TRABALHOS RELACIONADOS

Ao utilizar um algoritmo de classificação de aprendizado de máquina para classificar solicitações HTTP como normais ou anômalas é necessário extrair características dos dados em cada requisição e rotulá-los. Aprendendo como os recursos se relacionam ao rótulo, será produzido um modelo matemático que mapeia o relacionamento entre os recursos e os rótulos (ALTHUBITI; YUAN; ESTERLINE, 2017). Algoritmos de classificação baseada em *machine learning* têm presença marcante na literatura para detecções de anomalias. Apresenta-se a seguir os trabalhos que utilizaram aprendizado de máquina para classificação de ataques contra aplicações Web, especialmente aqueles com o uso de redes recorrentes.

2.1 TRABALHOS COM APRENDIZADO DE MÁQUINA TRADICIONAL E COMPARATIVOS

Alguns algoritmos precursores no aprendizado de máquina ainda são largamente utilizados na tarefa de detecção. Nesta seção destacam-se alguns trabalhos que utilizaram técnicas baseadas em aprendizado de máquina. Sahin e Sogukpınar (2017) utilizaram *K-Nearest Neighbors* (K-NN) e *Decision Tree* (DT) como classificadores. Os recursos para o modelo são extraídos a partir da seleção de palavras-chave resultando em 25 recursos. A árvore de decisão apresentou uma taxa de detecção de 96,26% e o conjunto de dados utilizado foi o CSIC 2010 (CSIC, 2010).

O trabalho de Mac et al. (2018) utilizou o Autoencoder, uma categoria de rede neural, para detectar padrões em solicitações HTTP/HTTPS. Uma das variantes testadas, o *Regularized Deep Autoencoder* (RDA), atingiu média de 5,1 milissegundos por requisição executando os experimentos em sistema Ubuntu 16.04 com processador Intel Core i5 e 8 GB de RAM. A precisão do modelo proposto com RDA foi de 94,64%, com taxa de detecção de 94,62% e a pontuação F-1 alcançou 94,63%. O conjunto de dados CSIC 2010 foi utilizado para os testes.

Zhang, Lu e Xu (2017) combinaram três técnicas sendo, Distribuição de Probabilidade, Cadeia de Markov Oculto e *Support Vector Machine* (SVM), onde os modelos dividem a avaliação de sete campos da requisição HTTP. Na proposta uma anomalia é acusada se pelo menos um deles relatar uma anomalia. As técnicas foram testadas em conjunto e individualmente, mas somente a detecção das técnicas agregadas teve desempenho satisfatório com taxa de detecção de 97,71% e taxa de falsos positivos de 2,02%. Os dados utilizados para o experimento são pedidos normais de rastreamentos de acesso da Wikipédia. No entanto, os autores não informaram dados referentes ao tempo de detecção. Epp, Funk e Cappo (2017) também utilizaram SVM como classificador de um *firewall* para aplicações da Web. O *One-Class SVM* utilizado nesta abordagem atingiu uma pontuação F-1 de 93% e uma taxa de detecção (ou True

Positive Rate como está no trabalho) 95%.

Ito e Iyatomi (2018) utilizaram uma rede neural convolucional ao nível de caractere (CLCNN) para extrair recurso das requisições HTTP e posterior classificação. Duas arquiteturas foram testadas sendo uma com apenas um segmento de convolução e *max pooling* e outra que paraleliza quatro segmentos deste. O conjunto de dados usado é o CSIC 2010. Os experimentos alcançaram uma acurácia de 98,8% com 2,35ms de tempo médio por requisição na arquitetura sem paralelização. A única informação sobre o computador utilizado nos experimentos foi a utilização de uma única GPU (Titan X pascal).

Dois trabalhos apresentaram um comparativo entre classificadores baseados em aprendizado de máquina. Althubiti, Yuan e Esterline (2017) compararam seis técnicas que incluem *Random Forest*, *Logistic Regression*, *AdaBoost*, *J48*, *Decision Tree* (CART e C4.5) e *Stochastic Gradient Descent* (SGD). Os autores usaram métodos de seleção de recursos com o software Weka¹ para classificar os cinco melhores recursos (dentre nove) para melhorar a precisão e diminuir o tempo de treinamento. Apesar de apresentar bons resultados preditivos em quase todas as técnicas testadas, a principal contribuição deste trabalho foi a redução dos recursos. Ao selecionar apenas 5 atributos para o modelo os autores conseguiram reduzir o tempo de treinamento.

Em outro trabalho comparativo Smitha, Hareesha e Kundapur (2019b) analisaram o potencial da ferramenta *Microsoft Azure Machine Learning Studio* (MAMLS) no uso de algoritmos de aprendizado de máquina. Os autores fazem uso do conjunto CSIC 2010 para testar a detecção de ataques como em uma aplicação Web. Os algoritmos *Logistic Regression* (LR), *Support Vector Machine* (SVM), *Decision Tree* (DT), e Redes Neurais Artificiais, além de variantes destes quatro, foram testados nos experimentos. Destaque para o SVM e LR que apresentaram as melhores pontuações *F1-score* com 93% e 96%, respectivamente.

2.2 TRABALHOS COM REDES RECORRENTES

Recentemente, as técnicas de aprendizado profundo vêm obtendo bons resultados e atraindo atenção da comunidade científica. Os modelos baseados em redes recorrentes têm sido cada vez mais pesquisados na detecção de anomalias. Bochem, Zhang e Hogrefe (2017) propõem uma abordagem baseada em rede neural de longo prazo (LSTM) para processar requisições HTTP ao nível de caractere. Um classificador binário atribui um valor de probabilidade a cada caractere da solicitação e depois multiplica esses valores para determinar a probabilidade geral da solicitação para defini-la como normal ou anômala. O ajuste de parâmetros da rede LSTM permite ao modelo a

¹ Weka é um software com uma biblioteca de algoritmos de aprendizado de máquina para tarefas de mineração de dados. O software foi desenvolvido sob licença GPL (General Public License) e contém ferramentas para pré-processamento, classificação, regressão, clustering, regras de associação e visualização de dados (HORNÍK; BUCHTA; ZEILEIS, 2009).

capacidade de escolher com flexibilidade entre diferentes taxas de detecção e falso positivo do classificador de acordo com a necessidade da aplicação.

Os autores Althubiti et al. (2018) também implementaram um modelo para detecção de intrusão para aplicações da Web usando redes neurais do tipo LSTM. Neste trabalho os pesquisadores buscaram encontrar os valores de parâmetros ideais para um melhor desempenho da rede neural utilizada no modelo. Também realizaram um comparativo entre dois otimizadores de redes neurais recorrentes com destaque para o Adam (KINGMA; BA, 2014). A rede neural recorrente recebeu na camada de entrada nove características extraídas das requisições HTTP do conjunto de dados CSIC 2010. O modelo obteve altos índices de acurácia, precisão e taxa de detecção, mas os autores não apresentam informações de custo computacional ou tempo de detecção.

Outra categoria de rede recorrente que vêm sendo explorada é a LSTM bidirecional. Em (HAO; LONG; YANG, 2019) os autores apresentam um método para detectar ataques Web com este tipo de rede. A abordagem proposta analisa requisições HTTP que são decodificadas e mapeadas para um vetor de palavras. Este vetor é posteriormente transformado em um vetor numérico para ser utilizado como entrada para a rede neural. A proposta apresenta apenas resultados relativos aos índices de detecção do modelo com um bom desempenho. O trabalho, porém, não apresenta informações sobre o *hardware* utilizado nos experimentos ou dados de tempo de detecção.

Outro trabalho a utilizar redes LSTM Bidirecionais foi apresentado por Yu et al. (2018). A proposta contempla um mecanismo de atenção que se trata de um recurso usado de maneira eficaz em processamento de linguagem natural e que auxilia na captura de informações essenciais para a detecção. O modelo foi comparado a uma rede neural convolucional, uma LSTM e a própria rede BI-LSTM isoladamente. O modelo obteve o melhor resultado nas métricas de precisão, taxa de detecção e medida F1. No entanto, a presença do mecanismo de atenção no modelo representa uma fase a mais em relação às demais soluções comparadas o que deve aumentar o tempo de detecção.

A rede recorrente conhecida como GRU (Gated Recurrent Unit) é uma variação com estrutura mais simples da LSTM. Zhao et al. (2018) compararam um modelo de *Random Forest* com outro baseado em rede GRU. Ambos os modelos utilizaram 21 recursos extraídos com base em análise léxica e estatística dos URLs e são usados para treinar e classificar URLs maliciosas. A GRU obtém a melhor desempenho entre os cenários testados com uma acurácia de 98,5% quando utiliza 240 mil URLs no conjunto de treino.

2.3 TRABALHOS COM EMPILHAMENTO DE TÉCNICAS

Na busca por melhor desempenho preditivo dos modelos de detecção alguns trabalhos de pesquisa utilizam redes recorrentes empilhadas com outras técnicas. Kim e Cho (2018) propuseram um método denominado C-LSTM que consiste na combinação de uma rede neural convolucional (CNN), uma memória de curto prazo (LSTM) e uma rede neural profunda (DNN) conectadas de maneira linear. A camada LSTM auxilia na identificação de recursos temporais. O conjunto de dados utilizado no trabalho tem uma proporção muito pequena de 0,02% de anomalias o que exigiu dos autores a utilização de um algoritmo de janela deslizante. Apesar de apresentar resultados individuais para LSTM, GRU e CNN, os resultados experimentais apontam a melhor acurácia para o modelo que agrupa as três redes com 98,5%.

Em (LIANG; ZHAO; YE, 2017c) as requisições HTTP são tokenizadas em duas abordagens diferentes: (i) a estrutura de parâmetros de consultas e (ii) a estrutura dos caminhos do URL. Duas redes neurais recorrentes paralelas processam os tokens e geram um vetor de probabilidade. Em seguida, uma rede neural *Multilayer Perceptron* (MLP) recebe este vetor e classifica a requisição. A vantagem desta abordagem é que ela é livre de seleção de recursos e pode ser personalizada para aprender padrões de solicitação normais para qualquer aplicativo da Web específico. No entanto, o modelo limitou-se às solicitações do tipo GET o que reduziu bastante a quantidade de dados utilizados para os experimentos.

Wang, Zhou e Chen (2018) exploraram a detecção de ataques na Web em redes convolucionais (CNN) e LSTM explorando seus desempenhos de maneira individual e combinadas. As experiências mostram que os modelos propostos podem ter um desempenho superior aos métodos tradicionais de classificação no conjunto de dados público CSIC com o modelo combinado alcançando pontuação F-1 de 98,9%.

2.4 RESUMO DOS TRABALHOS

A tabela 1 faz uma compilação dos trabalhos relacionados com dados de desempenho preditivo, técnicas e conjunto de dados utilizados.

Algumas considerações devem ser feitas quanto aos trabalhos listados nesta seção. O desempenho apresentado pelas redes recorrentes LSTM é expressivo em vários deles, apesar de que, alguns resultados com acurácia muito próxima de 99,9 podem indicar que o modelo tenha se adaptado demais ao conjunto de dados. Nenhum deles apresenta informações detalhadas sobre o processo de escolha dos parâmetros de configurações da rede neural.

O conjunto de dados CSIC 2010 (CSIC, 2010) é utilizado em onze de quinze trabalhos citados. É uma vantagem, pois, permite a comparação de resultados entre os trabalhos. Porém, o conjunto possui uma versão mais recente do ano de 2012

(TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2012) com ataques rotulados e dados anômalos que não são ataques. Trabalhar com este conjunto de dados permite avaliar a reação do modelo a cada tipo de ataque.

Há pouca informação referente ao custo computacional ou tempo de treinamento e de detecção dos modelos implementados. Somente dois trabalhos informam um tempo médio de detecção do modelo. A falta de informação nesse sentido não permite que se faça uma avaliação do equilíbrio do modelo entre eficiência e eficácia.

Tabela 1 – Trabalhos relacionados - Métricas de Desempenho predito utilizadas: ACC=Acurácia, TD=Taxa de Detecção, TFP=Taxa de Falsos Positivos, F1=Medida F1-Score, P=Precisão.

Ano	Título	Modelo / Técnica	Conjunto de Dados	Métricas	Tempo de detecção
2017	Streamlined anomaly detection in web requests using recurrent neural networks	LSTM	CSIC 2010	Best: LSTM TD: 97% TFP: 4%	-
2018	Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection	LSTM	CSIC 2010	ACC: 99.9% TD: 99.5% P: 99.5%	-
2019	BL-IDS: Detecting Web Attacks Using Bi-LSTM Model Based on Deep Learning	BI-LSTM	CSIC 2010	TD: 98.1% P: 98.3% F1: 0.985	-
2018	Attention-Based Bi-LSTM Model for Anomalous HTTP Traffic Detection	BI-LSTM	Dados próprios	Não informados	-
2019	Classifying Malicious URLs Using Gated Recurrent Neural Networks	GRU	Chinese Internet security company	Best: GRU ACC: 98.5%	-

Ano	Título	Modelo / Técnica	Conjunto de Dados	Métricas	Tempo de detecção
2018	Web traffic anomaly detection using C-LSTM neural networks	CNN-LSTM-DNN	Webscope S5 do Yahoo	ACC: 98.6% P: 96.2% TD: 89,7% F1: 92,3	-
2017	Anomaly-Based Web Attack Detection: A Deep Learning Approach	LSTM / GRU / RNN	CSIC 2010	Best: GRU ACC: 98.5% TD: 98.80%	-
2018	Evaluating CNN and LSTM for Web Attack Detection	CNN-LSTM	CSIC 2010	P: 98.9% TD: 98.8% F1: 0.989	-
2018	Web Application Firewall using Character-level Convolutional Neural Network	CLCNN	CSIC 2010	ACC: 98.8%	2,35 ms - Hardware não informado
2018	Detecting Attacks on Web Applications using Autoencoder	Autoencoder (RDA)	CSIC 2010	P: 94.6% TD: 94.6% F-1: 0.946	5.1 ms / Ubuntu 16.04 - Intel Core i5 - 8 GB RAM
2017	An efficient firewall for web applications (EFWA)	DT / KNN	CSIC 2010	Best: DT ACC: 96.2% P: 96.3% TD: 96.3% F1: 0.963	-
2017	An Anomaly Detection Method Based on Multi-models to Detect Web Attacks	PDM-HM-On-Class SVM	Pedidos de rastreamentos de acesso da Wikipedia	ACC: 97.7% TFP: 2.02%	-

Ano	Título	Modelo / Técnica	Conjunto de Dados	Métricas	Tempo de detecção
2017	Anomaly-based Web Application Firewall using HTTP-specific features and One-Class SVM	One-Class SVM	CSIC 2010	TD: 95.0% F1: 0.93	-
2017	Analyzing HTTP requests for web intrusion detection	Comparativo entre vários	CSIC 2010	Best: RF, LR, ABc, SGDc F1: 0.999	-
2019	A Machine Learning Approach for Web Intrusion Detection: MAMLS Perspective	Comparativo entre vários	CSIC 2010	Best: LR ACC: 97% P: 92.0% TD: 95.0% F1: 0.96	-

Fonte: Levantamento realizado pelo autor.

A proposta deste trabalho consiste na experimentação de um modelo que utilize redes neurais recorrentes, mas que mantenha um equilíbrio entre o desempenho preditivo e o tempo de detecção permitindo que a sua atuação em um servidor Web seja discreta e escalável.

3 FUNDAMENTAÇÃO TEÓRICA

Para uma melhor compreensão das atividades desenvolvidas neste trabalho são apresentados a seguir conceitos importantes. A seção 3.1 apresenta as principais categorias de ataques contra aplicações da Web, taxonomia e especificidades de cada tipo. Em seguida, na seção 3.2, apresentam-se os conceitos de Redes Neurais Recorrentes e os princípios de suas principais vertentes abordadas neste trabalho. Por fim, a seção 3.3 aborda o filtro de Bloom, usado como primeiro estágio do modelo de detecção deste trabalho.

3.1 ATAQUES CONTRA APLICAÇÕES WEB

A Web se define como um ambiente de comunicação com arquitetura cliente-servidor em que as partes trocam informações utilizando o protocolo HTTP - *Hyper Text Transfer Protocol* ou HTTPS (com criptografia) (CAPPO, 2017). Uma aplicação web consiste em um número finito de páginas que podem ser estáticas mas, em sua maioria, são dinâmicas. Páginas dinâmicas são aquelas que recuperam seu conteúdo de um banco de dados sob demanda (GROVE, 2009). Isso significa que as aplicações consultam o servidor de conteúdo e permitem a visualização de informações dinamicamente de acordo com a solicitação do usuário.

O protocolo HTTP é baseado na troca de mensagens entre o navegador do cliente e o servidor da Web. As mensagens de solicitação e resposta consistem em cabeçalhos que contêm informações como método, recurso, versão do protocolo, cabeçalhos, argumentos e, em alguns casos, o corpo das solicitações. “Essa arquitetura é robusta, flexível e permite fácil troca de informações entre usuários da Internet. Isso explica por que de seu grande desenvolvimento, mas, ao mesmo tempo, porque é um dos principais objetivos dos cibercriminosos.” (CAPPO, 2017).

Segundo relatório da Symantec (SYMANTEC, Fevereiro/2019), em 2018, um em cada dez URLs analisados foram identificados como maliciosos. Estes números representaram um aumento em comparação com o ano anterior em que essa relação era de uma URL maliciosa em cada dezesseis. Além disso, a Symantec bloqueou mais de 1,3 milhão de ataques *web* diariamente e o número global de ameaças registradas em seus produtos aumentaram 56% em 2018. Em seu relatório mais recente a OWASP (2017) aponta ameaças que agem sobre a manipulação dos URLs como os ataques de *Code Injection*, *Buffer Overflow*, *Cross-Site Scripting* ou *Directory Traversal*, entre outros. Estes dados são corroborados por outro relatório da Technologies (2019) que apresenta as vulnerabilidades mais comuns registradas em 2018.

A manipulação de argumentos de entrada nas requisições é a essência dos ataques de injeção de código. Falhas de injeção ocorrem quando um invasor pode enviar dados hostis a um interpretador. As entradas do usuário são tratadas como

entidades lexicais isoladas que, se não forem adequadamente tratadas, podem fazer com que o aplicativo da Web gere saída não intencional, o que configura um ataque de injeção de comando (SU; WASSERMANN, 2006). Uma característica destes ataques é que eles perturbam de alguma forma a construção da requisição normal, conforme apresentado por (CAPPO, 2017). Alguns exemplos são apresentados a seguir.

SQL Injection (SQLi): O ataque consiste na inserção de código malicioso na consulta SQL em uma aplicação no momento de sua comunicação com um banco de dados, permitindo ao atacante obter acesso ou modificar dados para os quais não possui privilégios (SKARUZ; SEREDYNSKI, 2007). O invasor pode alterar a estrutura da consulta e ir escalando privilégios no servidor Web. Na Figura 1, a expressão `1=1` será sempre verdadeira e a inserção de dois traços comenta o que vem a seguir, excluindo o campo `'password'` da consulta.

Figura 1 – Exemplo de requisição HTTP com injeção de código malicioso na consulta SQL.

```
$query="select * from users where login=' ' or 1=1 --' and password=' ' "
```

Fonte: Próprio autor

BufferOverflow: Nesta categoria de ataque o invasor tenta modificar o estado da memória do programa, fornecendo dados de entrada em excesso. Com isso o tamanho do buffer de memória copiaria o excesso de dados para locais adjacentes. Ao controlar o conteúdo, o invasor pode fazer com que o programa se desvie do objetivo pretendido (RUWASE; LAM, 2004). A figura 2 traz um exemplo deste ataque. A presença de um caractere “X” repetido várias vezes com objetivo de apenas consumir memória. Isso altera de maneira significativa o tamanho da requisição e a distribuição normal da frequência dos caracteres.

Figura 2 – Exemplo de inserção de caracteres aleatórios em ataque de Estouro de Buffer.

```
GET /default.ida?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX(...)
```

Fonte: Próprio autor

Cross Site Scripting (XSS): Tem objetivo de injetar rotinas (*scripts*) no código de execução. Os invasores exploram a relação de confiança entre um servidor da Web e um navegador. É mais um exemplo de ataque que pode se apropriar da conexão de um usuário com privilégios no sistema através da inserção de dados maliciosos em uma requisição (SKARUZ; SEREDYNSKI, 2007). Na situação exemplificada por (CAPPO, 2017) mostrada na figura 3 a requisição sofre uma alteração tendo recebido um `<script>... </script>` no lugar onde deveria receber um nome de usuário. O código

malicioso modifica o atributo *document.location* para enviar dados privados ao sítio web do invasor.

Figura 3 – Exemplo de um ataque XSS com a injeção de um script malicioso.

```
GET /index.php?sessionid=12312312&username=<script>document.location='http://attackerhost(...)</script>
```

Fonte: Próprio autor

Server-Side Include são diretrizes presentes em aplicações da Web usadas para alimentar uma página HTML que possua conteúdo dinâmico, sendo que, o servidor da web analisa o SSI antes de fornecer a página ao usuário. SSIs usadas para executar algumas ações antes do carregamento da página atual ou enquanto a página está sendo visualizada (OWASP,). O ataque de Injeção SSI (SSI) permite explorar dados de uma aplicação Web injetando *scripts* em páginas HTML ou executando códigos arbitrários remotamente. O ataque será bem-sucedido se o servidor da Web permitir a execução do SSI sem a validação adequada. No exemplo apresentado na Figura 4 um *script* configura para ambiente Linux que renomeia o arquivo *shell.txt* para a extensão *.php*.

Figura 4 – Exemplo de um script que pode ser inserido em requisição para ataque de Injeção SSI

```
<!--#exec cmd="wget http://mysite.com/shell.txt | rename shell.txt shell.php" -->
```

Além destes, pode-se citar como ataques que alteram a composição normal de uma requisição outras categorias como o *XML Path Language injection* (XPath) que ocorre em consultas do tipo *XML Path Language* ou a *Lightweight Directory Access Protocol Injection* (LDAPi) que é um ataque semelhante, mas aplicadas em sentenças LDAP (CAPPO, 2017). Cita-se ainda a Inclusão de Arquivos (*File Inclusion*) quando um atacante consegue incluir um arquivo remoto como se fosse documento local burlando as suas permissões. Segundo Watson (2007), isto acontece porque aplicação confia nos arquivos de entrada enviados pelo usuário ou nas referências a objetos externos e executa o conteúdo malicioso.

A variação das requisições quando afetadas por ataques de injeção pode ser percebida em variados aspectos. Em alguns casos, como no estouro de buffer, o tamanho da requisição sofre uma variação considerável em seu comprimento e quantidade de caracteres. Em outros, a presença de elementos específicos podem representar uma possível anomalia como a marcação *<script>* no ataque XSS. Percebe-se que mesmo restringindo a abordagem a ataques de injeção de comando o seu tratamento é bastante complexo, o que amplia o desafio na detecção de anomalias contra aplicações da Web.

3.2 REDES NEURAIIS RECORRENTES

No campo do aprendizado de máquina existem situações em que o contexto é fundamental para o aprendizado. Redes Neurais Recorrentes (Recurrent Neural Networks - RNNs) são um tipo de rede neural artificial, inseridas no campo do aprendizado profundo. São projetadas para reconhecer padrões em sequências de dados, como texto, dados de sensores, ou informações em série que estabeleçam um ordenamento lógico (LE; KIM; KIM, 2017). RNNs consideram em seus algoritmos o tempo e a sequência, especialmente quando possuem uma dimensão temporal.

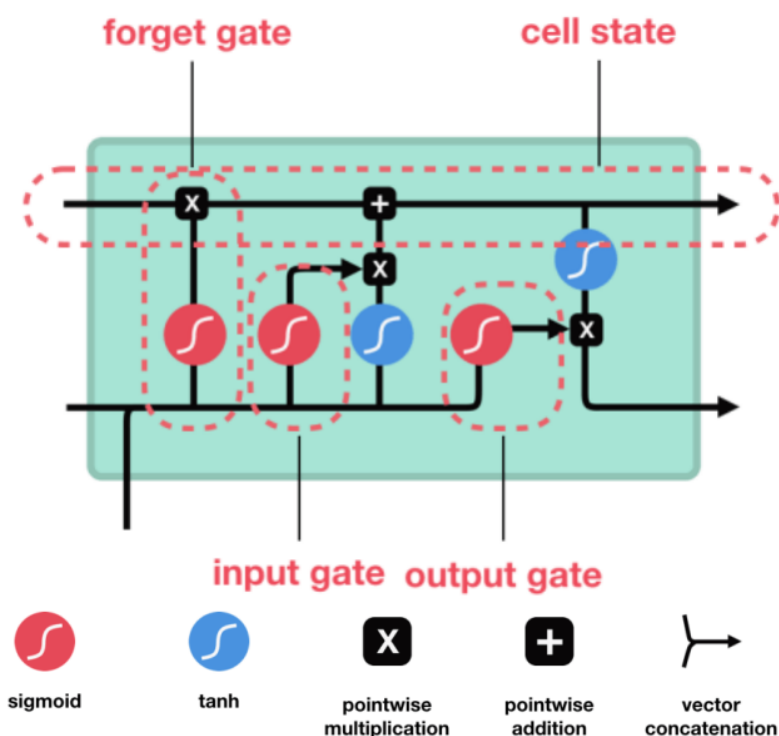
Nas próximas sub-seções são apresentadas três modelos de redes neurais recorrentes que foram utilizadas como detector do modelo proposto neste trabalho.

3.2.1 Long Short Term Memory.

Em algumas ocasiões, as RNNs podem fazer uso de informações em sequências extremamente longas. Os valores do gradiente encolherão ou aumentarão exponencialmente à medida que se propaga a cada etapa do tempo. Desta forma, na prática, a rede limita-se a recuperar informação para trás apenas alguns poucos passos. Este problema é conhecido como desaparecimento do gradiente (*Vanish Gradient*). Para superar este problema Hochreiter e Schmidhuber (1997) propuseram um conceito de célula de memória de longo prazo (LSTM).

O objetivo principal do LSTM é alcançar a descida do gradiente que desaparece e evitar problemas de dependência a longo prazo (ALTHUBITI; JONES; ROY, 2018). Elas possuem uma estrutura que lhes permite memorizar ou descartar informações para prever o comportamento dos próximo intervalo de tempo (timestep).

Conforme ilustrado pela Figura 5, a célula LSTM é composta por três “portões” que controlam o fluxo de informações e decidem o que é importante memorizar. O mais a esquerda, chamado de portão de esquecimento (*forget gate*), decide sobre que detalhes podem ser descartados da memória. O do centro é o portão de entrada (*input gate*) e analisa a nova entrada com objetivo de avaliar se a memória deve ser alterada. Finalmente, o da direita, denominado portão de saída (*output gate*), captura informação do estado da célula para gerar uma saída.

Figura 5 – Célula *Long Short-Term Memory* - LSTM

Fonte: (DATA SCIENCE ACADEMY, 2019)

Por sua natureza recorrente as células LSTM recebem sempre duas entradas que representam o presente e o passado recente. Numa delas chega a informação do estado atual da célula. A outra recebe a informação gerada pela saída da célula anterior.

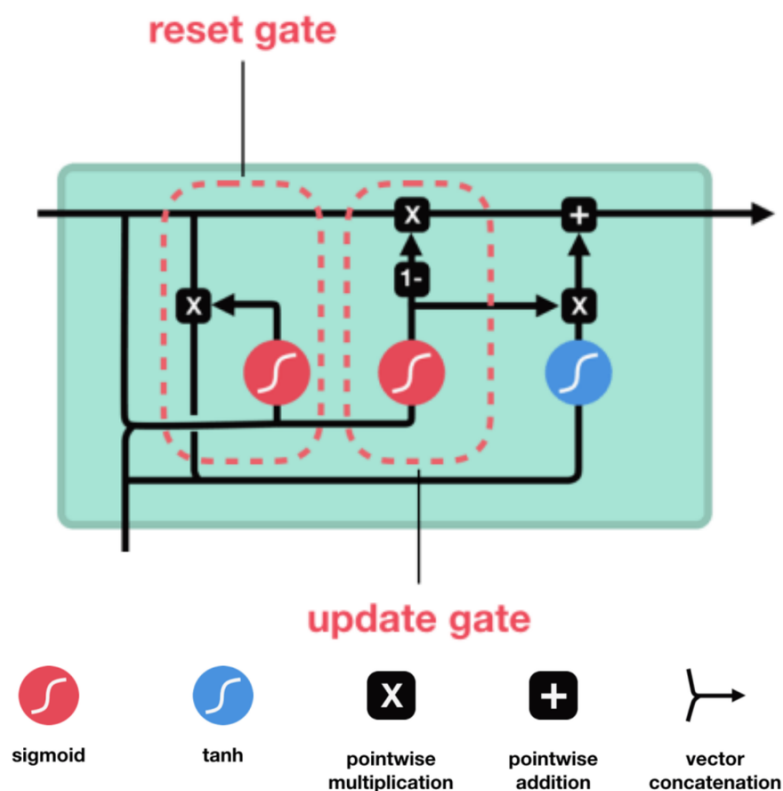
Falando didaticamente, ter memória é um bom recurso para o aprendizado da rede neural recorrente, no entanto, saber o que deve ser memorizado é fundamental para o desempenho. Se a aplicação pretende, por exemplo, prever a próxima palavra em uma sequência de 10 palavras, a rede pode esquecer artigos no meio da sentença que não influenciam para a previsão. Saber o que memorizar é tão importante quanto saber o que esquecer. Deste modo, configurar adequadamente a rede LSTM pode ser uma tarefa complexa.

3.2.2 Gated Recurrent Unit

A Unidade Recorrente Fechada (Gated Recurrent Unit - GRU) foi criada para capturar dependências em vários intervalos de tempo. É semelhante ao LSTM, mas tem uma estrutura mais simples. Comparado ao LSTM, a GRU não possui um estado de célula (*cell state*) separado, nem o portão de saída (*output gate*). A Figura 6 apresenta a estrutura da célula GRU onde é possível observar a presença do portão de atualização (*update gate*) e o portão de redefinição (*reset gate*). O primeiro funciona como combinação dos portões de entrada e esquecimento do LSTM. O último é

responsável por decidir quais informações devem ser usadas para compor a ativação do candidato. Da mesma forma, as entradas da célula GRU na etapa de tempo são a ativação da célula GRU da camada inferior e a ativação da célula GRU da etapa de tempo anterior. (DATA SCIENCE ACADEMY, 2019).

Figura 6 – Célula *Gated Recurrent Unit* - GRU



Fonte: (DATA SCIENCE ACADEMY, 2019)

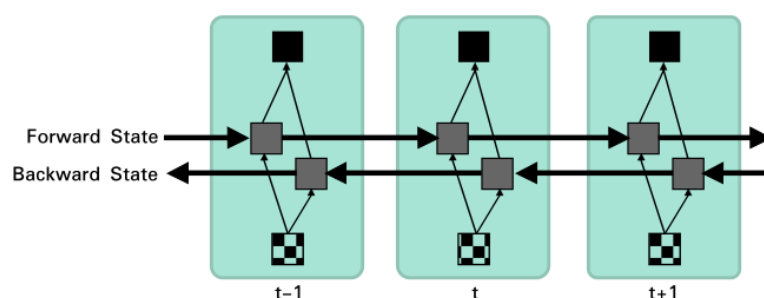
Embora os modelos LSTM e GRU permitam um aprendizado bem-sucedido nas RNNs, eles introduzem um aumento na parametrização através de seus portões. Conseqüentemente, há uma despesa computacional adicionada em relação ao modelo simples de RNN. A rede neural LSTM emprega três redes de portas distintas e a GRU duas redes de portas.

3.2.3 LSTM Bidirecional

O LSTM Bidirecional também vem sendo bastante utilizado na defesa de aplicações da Web (YU et al., 2018; LIANG; DENG; CUI, 2019; HAO; LONG; YANG, 2019). Este é um recurso muito utilizado no reconhecimento de fala, onde frases inteiras são transcritas de uma só vez, sendo possível explorar o contexto futuro também (GRAVES; JAITLEY; MOHAMED, 2013). Ao adotar a rede bidirecional Hao, Long e Yang (2019) consideram a possibilidade de de treinar a rede com informações do passado e do futuro.

Na célula Bidirecional LSTM os recursos são processados simultaneamente em dois sentidos como apresentando na Figura 7. Enquanto a entrada é processada no sentido direto em uma rede (*Forward State*), a mesma entrada é processada em sentido inverso (*backward state*) na outra.

Figura 7 – Célula LSTM Bidirecional - BI-LSTM



Fonte: Próprio autor (baseado em HAO, LONG e YANG 2019).

A saída das duas redes produzidas na fase bidirecional são concatenadas e seus valores normalizados em uma função de ativação produzindo a saída final da rede (LIANG; DENG; CUI, 2019).

3.2.4 Configuração de Redes Neurais Recorrentes

A configuração de redes neurais é uma tarefa complexa que envolve diversas variáveis que devem ser definidas adequadamente com o intuito de maximizar o seu desempenho. Os hiperparâmetros são usados para configurar vários aspectos do algoritmo utilizado e influenciam de variadas formas na rede neural e em seu desempenho (CLAESEN; MOOR, 2015). A escolha de hiperparâmetros pode representar a diferença entre o desempenho pífio ou excelente (REIMERS; GUREVYCH, 2017).

Em redes neurais é possível ajustar parâmetros como o número de unidades de células recorrentes (*Units*). Esta variável define a profundidade da rede neural. Redes recorrentes estão inseridas no campo do aprendizado profundo e podem trabalhar com um grande número de camadas. A profundidade maior, no entanto, eleva o tempo de treinamento das redes.

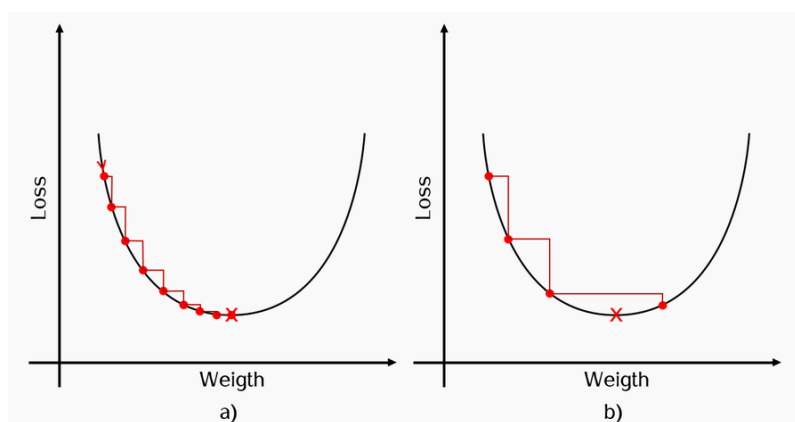
Também é possível inserir recursos como a taxa de esquecimento (*Dropout*) que define aleatoriamente uma porção das entradas como 0 para evitar o excesso de ajustes.

Há ainda ajustes que podem ser feitos na fase de treinamento das redes neurais como o tamanho do lote e o número de épocas. O tamanho do lote (*Batch Size*) é uma otimização nesta fase de treinamento que consiste no número de padrões mostrados na rede antes da atualização dos pesos. O número de épocas (*Epochs*) é o número de vezes que deseja-se mostrar o conjunto de dados para a rede durante o treinamento.

Os algoritmos de otimização (*Optimizer*) baseados em descida de gradiente são bastante populares na implementação de redes neurais. Eles são responsáveis pela minimização da função objetivo do sistema neural (REIMERS; GUREVYCH, 2017). Em testes realizados por Reimers e Gurevych (2017) os otimizadores *Adam*, *Nadam* e *RMSprop* produziram resultados mais estáveis e melhores para as tarefas de rotulagem de sequência. Estes otimizadores costumam ser utilizados em redes recorrentes para detecção de anomalias como em (KINGMA; BA, 2014).

A taxa de aprendizado é um hiperparâmetro presente nos otimizadores que controla o ajuste dinâmico dos pesos (*weight*) da rede neural em relação ao gradiente de perda (*loss*). Ela determina o tamanho das etapas para atingir um mínimo local. Valores muito pequenos representam uma viagem mais lenta ao longo da inclinação descendente. Enquanto, valores altos podem implicar na perda do mínimo global (ZULKIFLI, 2018). Como ilustrado na Figura 8a) com uma taxa de aprendizagem pequena a descida pelo gradiente pode ser lenta, mas tende a ser mais preciso na busca por um mínimo local. Na Figura 8b) com uma taxa maior é possível que a descida do gradiente falhe na convergência e ultrapasse o mínimo local.

Figura 8 – Descida do gradiente com a) uma taxa de aprendizagem pequena e b) taxa de aprendizagem grande



Fonte: Próprio autor (baseado em ZULKIFLI, 2018)

Todos esses ajustes podem ser feitos através da experimentação individual dos valores. Isso requer maior tempo e experiência do pesquisador, porém, é possível automatizar uma tarefa de testes com várias combinações e verificar a melhor configuração através de uma pontuação. Além disso, a escolha do tipo de célula recorrente ideal acaba sendo um elemento a mais na configuração da rede neural. A hiperparâmetrização das redes neurais utilizadas neste trabalho foi explorada no Capítulo 6.

3.3 FILTRO DE BLOOM

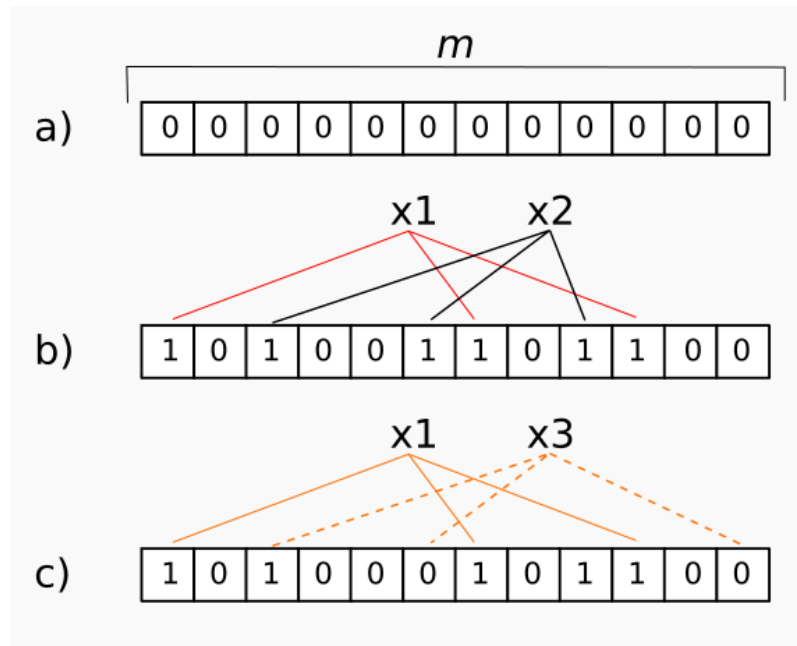
O Filtro de Bloom (BLOOM, 1970) é uma estrutura de dados simples e com espaço eficiente para representar um conjunto e oferecer suporte a consultas de associação. Tal filtro permite testar se um determinado elemento pertence a um conjunto de forma probabilística. Os resultados possíveis para uma consulta ao filtro é que determinado elemento consultado pode pertencer ao conjunto ou definitivamente não pertence ao conjunto. Isto por que, ao realizar uma consulta no filtro o resultado negativo é taxativo. O elemento com certeza não faz parte do conjunto. Por outro lado, se o resultado for positivo isso indica que é possível que o elemento faça parte do conjunto, porém, há uma pequena possibilidade deste resultado ser proveniente de uma colisão.

A eficiência na utilização dos filtros de Bloom dependerá de três parâmetros. São eles: (1) tamanho do filtro; (2) número de funções hash utilizadas no filtro; (3) número de elementos adicionados ao conjunto. Basicamente, pode-se reduzir as chances de falsos positivos de duas formas: alterar o número de funções hash (h) ou aumentar o tamanho do filtro de Bloom (m) (CRUZ, 2012, p. 49).

Para representar um determinado conjunto de elementos $S = \{x_1, x_2, \dots, x_n\}$ o filtro é descrito como um vetor de tamanho m de posições inicialmente preenchidos com valor 0. Cada elemento inserido no vetor de bits passa por um número k de funções de *hash* independentes. Estas funções mapeiam as posições no vetor que vão representar este elemento e alteram o valor para 1 nestas posições (figura 9b). Para verificar se um item está no vetor, aplica-se a função *hash* e verifica se todas as posições do elemento estão configuradas para o valor 1. Se todas as posições apontadas pelos *hash* estiverem definidas como 1, assume-se então que o elemento consultado pode estar no conjunto. Caso contrário, o elemento y definitivamente não é um membro do conjunto (BRODER; MITZENMACHER, 2004).

A figura 9 ilustra a dinâmica de inserção e consulta de elementos no Filtro de Bloom. Na figura 9a) um filtro de tamanho m é inicializado com todos os valores zerados. Na figura 9b) são aplicadas três funções *hash* a cada elemento inserido, x_1 e x_2 . Cada *hash* aplicada aponta para uma posição no filtro e o valor nessa posição é alterado para 1. Na figura 9c) dois elementos são consultados. O elemento x_1 , que havia sido inserido anteriormente, ao passar pelas funções *hash*, apontará para as mesmas posições, o que indica que ele faz parte do conjunto presente no filtro. O elemento x_3 , ao ser consultado, aponta para posições ainda com valor zero, o que indica que ele não parte do conjunto.

Figura 9 – (a) Filtro vazio (b) Preenchimento do filtro (c) Consulta de elementos no filtro.



Fonte: Próprio autor

Há uma compensação (*trade-off*) entre os requisitos de memória do filtro de Bloom e a taxa de falsos positivos, neste trabalho tratado sempre pelo termo colisões¹. Esta taxa pode ser ajustada através da escolha dos parâmetros m e k . Se a quantidade de elementos (variável n) que se pretende usar para preencher o vetor Bloom for conhecida é possível calcular o tamanho do vetor de bits do filtro, inclusive, determinando a taxa de probabilidade de colisões P (equação 3.1). Conhecendo o tamanho do vetor e a quantidade de itens é possível calcular o número ideal de funções hash utilizadas para as operações de inserção e consulta de elementos do filtro (equação 3.2).

$$m = -\frac{n \ln(P)}{(\ln 2)^2} \quad (3.1)$$

$$k = \frac{m}{n} \ln(2) \quad (3.2)$$

Observa-se que o filtro Bloom tem a vantagem ser mais eficiente em espaço e o tempo de consulta são inferiores em comparação aos algoritmos comuns (JIANG et al., 2018), diferenças essenciais para aplicações da Web.

¹ A literatura costuma chamar a colisão no Filtro de Bloom de falso positivo. No entanto, para não haver confusão com o termo usado para representar um falso positivo do modelo, neste trabalho, utilizou-se apenas o termo colisão.

4 MODELO FILTRO DE BLOOM COM REDE NEURAL RECORRENTE (BLOOM-RNN)

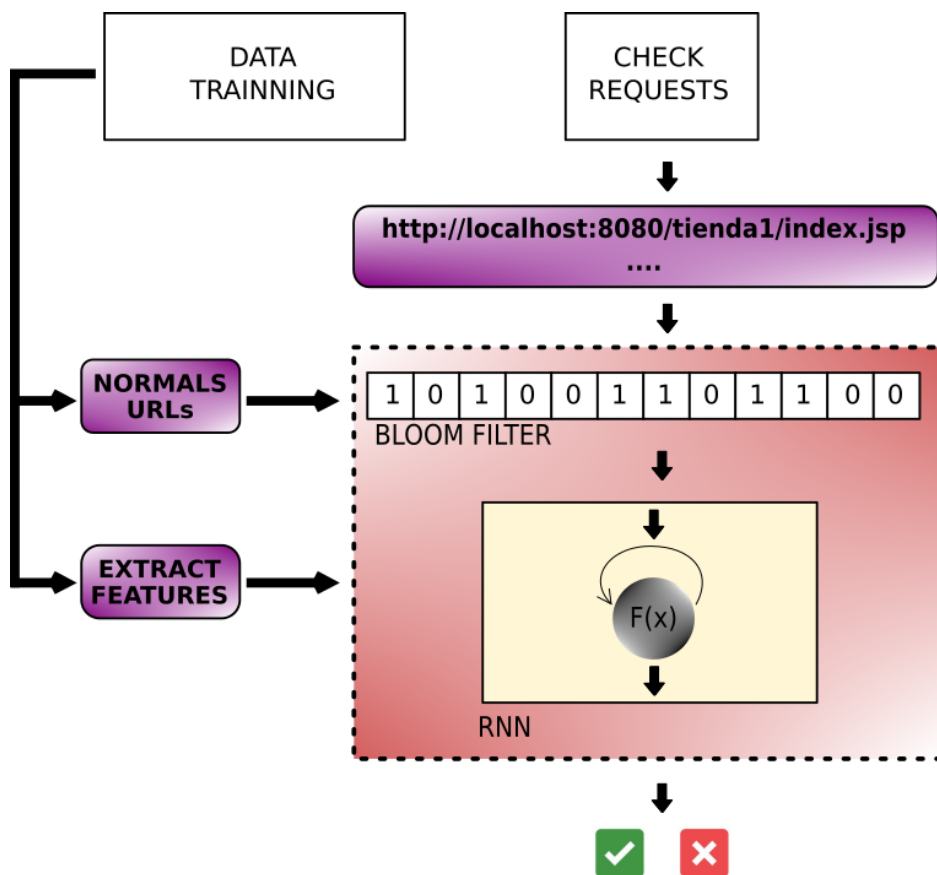
A proposta apresentada neste trabalho consiste num modelo de detecção de ataques contra aplicações Web em dois estágios. No primeiro, o Filtro de Bloom é alimentado com amostras de requisições HTTP de padrões normais e funciona como lista de assinaturas previamente conhecidas. No segundo estágio, a Rede Neural Recorrente atua como classificador analisando vetores numéricos que representam as requisições HTTP. A implementação do modelo considerou, nas duas fases, a redução da dimensionalidade dos dados como fator de eficiência e, especificamente, a exploração de variantes da rede neural recorrente para incrementar a acurácia preditiva.

Uma descrição sucinta do processo de aprendizado e detecção no modelo proposto é realizada na seção 4.1. Nas seções que seguem apresenta-se o detalhamento do pré-processamento que extrai das requisições as características de interesse (seção 4.2), do estágio de redução de dimensionalidade com o Filtro de Bloom (4.3) e do estágio de detecção com a Rede Neural Recorrente (4.4).

4.1 FLUXO DE INFORMAÇÕES NO MODELO DE DETECÇÃO

Em modelos baseados em aprendizado de máquina a informação costuma passar por 3 fases, sendo, pré-processamento, aprendizado e avaliação dos dados (ZHOU et al., 2017). Portanto, para que o modelo seja funcional é necessário haver amostras de requisições da aplicação Web que se pretende proteger. É o que se chama de dados de treinamento. Para o modelo proposto neste trabalho estes dados servem para treinar a rede neural recorrente, mas também alimentam o filtro de Bloom. A diferença é que no filtro apenas as requisições normais são usadas, enquanto para treinar a rede neural se utilizam dados malignos e benignos. A figura 10 ilustra os módulos do modelo proposto e o fluxo da informação. Os dados de treinamento alimentam o módulo de URLs normais (*normals URLs*) que preenchem o filtro. O módulo de extração de recursos (*extract features*) transforma os URLs de elemento textual para vetor numérico. Com o Bloom preenchido, e a rede neural treinada, o módulo está apto para realizar detecções.

Figura 10 – Fluxo da informação no modelo de detecção



Fonte: Próprio autor

Na fase de verificação as requisições são checadas individualmente, primeiro, com uma consulta no filtro. A identificação de uma requisição semelhante aponta para um URL previamente conhecida, portanto, ela pode ser caracterizada como normal. Caso contrário a requisição é enviada ao módulo de extração de recursos, pois, a verificação na rede neural sempre será a partir de vetor de números. Só então a requisição é enviada ao detector (*RNN*) que fará a classificação e atestará se a requisição se trata de um ataque.

4.2 PRÉ-PROCESSAMENTO DOS DADOS

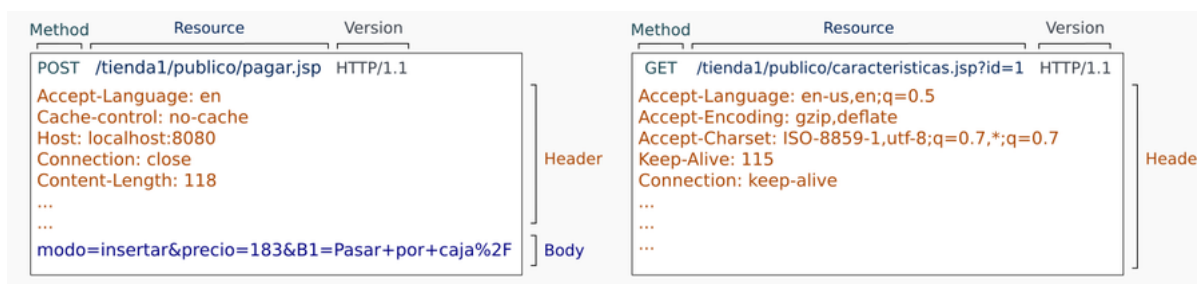
O pré-processamento dos dados compreende duas etapas. Num primeiro momento as requisições são tratadas para remover dados não utilizados pelo modelo de detecção. O filtro opera com as *strings* resultantes do processo de limpeza. Depois, as requisições que se direcionam a rede neural passam por um processo de extração de atributos numéricos. Estas atividades são detalhadas a seguir.

4.2.1 Limpeza dos dados

Neste trabalho, o modelo de detecção utiliza apenas o campo recurso (resource) da requisição HTTP. Isto vale para requisições com métodos dos tipos GET e DELETE. Os métodos dos tipos POST, PUT e PATCH possuem uma construção diferente. Com esses é possível enviar parâmetros no URL e no corpo da mensagem de solicitação (CAPPO, 2017).

A figura 11 apresenta como exemplo a estrutura de duas requisições. À esquerda, a solicitação HTTP do método (*method*) POST apresenta o URL do aplicativo (*resource*) e os parâmetros estão no corpo (*body*) da mensagem HTTP. À direita, a imagem apresenta uma solicitação do método GET, da qual toda informação necessária para utilização no modelo está no recurso (*resource*). Nesta etapa do pré-processamento os dados da versão (*version*) e do núcleo do cabeçalho (*header*) são descartados.

Figura 11 – Exemplos de requisições dos tipos POST e GET



Fonte: Próprio autor

O tratamento das requisições faz a limpeza de informações desnecessárias para o processo de detecção de anomalias. A *string* que é utilizada para classificação no modelo BLOOM-RNN consiste no campo *resource* quando se tratar de requisições dos métodos GET e DELETE. Para requisições dos métodos POST, PUT e PATCH a *string* é composta pela concatenação dos campos *resource* e *body*. Somente as *strings* provenientes do conjunto de requisições normais são usadas para preencher o Filtro de Bloom.

Para que possam ser processadas pela rede neural as strings correspondentes aos URLs passam pela seleção de recursos que as transforma em vetor numérico. Este processo é apresentado na seção a seguir.

4.2.2 Seleção de Recursos

A seleção de recursos é uma estratégia para redução da dimensionalidade dos dados (HERRERA-SEMENETS et al., 2018). Trata-se do processo de escolha dos atributos mais relevantes para representar os dados em vetores numéricos. Esses

atributos auxiliam na detecção de ataques. A escolha dos atributos mais relevantes têm sido alvo de pesquisas (NGUYEN et al., 2011; ALTHUBITI; YUAN; ESTERLINE, 2017; GIMÉNEZ et al., 2015).

Nguyen et al. (2011) elencaram trinta recursos que consideraram relevantes para o processo de detecção a partir do conhecimento especializado e aplicaram a medida de seleção de recurso genérico (GeFS). Usando a medida CFS (*Correlation Feature Selection*), uma instância da GeFS, os autores removeram atributos redundantes e irrelevantes para gerar um conjunto de dados reduzido. Esse processo reduziu consideravelmente os atributos (de trinta para nove) com pouco impacto na precisão da detecção (0,12% menor). Em outro trabalho Althubiti et al. (2018) aplicou estes nove atributos para gerar vetores que foram usados na alimentação de uma rede neural recorrente e que resultou em bom desempenho preditivo. Como a seleção de recursos não é o foco deste trabalho, adota-se o conjunto reduzido de recursos proposto por Mac et al. (2018) e adotado por Althubiti et al. (2018). Os atributos selecionados são descritos na tabela 2.

Tabela 2 – Exemplo de uma requisição e os valores extraídos a partir da seleção de atributos

Atributos	
1	Length of the request
2	Length of the path
3	Number of letter chars in the path
4	Number of 'special' chars in the path
5	Length of the arguments
6	Number of arguments
7	Number of letters in the arguments
8	Number of digits in the arguments
9	Maximum byte value in the request

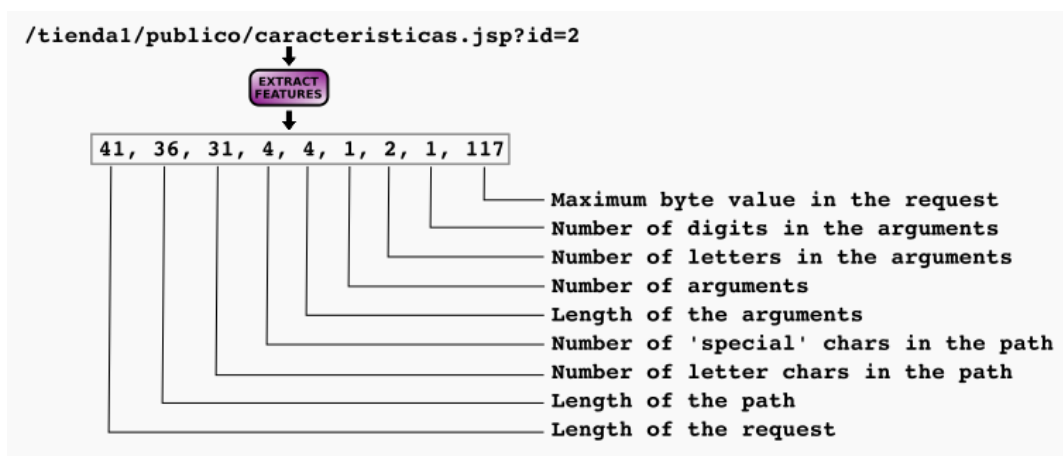
Fonte: (ALTHUBITI, S. 2018, p. 3)

O conjunto reduzido de atributos selecionados por Nguyen et al. (2011) considera, basicamente, a contagem de caracteres (atributos 3, 4, 6, 7, 8) e tamanho de determinadas partes da requisição (atributos 1, 2, 5). Para Giménez et al. (2015) nem todos os caracteres têm a mesma importância para ataques Web. Caracteres especiais, por exemplo, possuem grande relevância na detecção de vários ataques. Por isso, alguns atributos consideram a contagem, separadamente, de letras, dígitos

e caracteres especiais não alfanuméricos. As medidas de comprimento, no que lhe concerne, são um critério útil para detectar ataques, pois, os valores de solicitações normais geralmente contêm poucos bytes, enquanto, muitos ataques na Web usam uma quantidade expressiva de caracteres de entrada (GIMÉNEZ et al., 2015). Há um único atributo que difere das medidas de quantidade e tamanho. O atributo *Maximum Byte Value* (atributo 9) é a representação numérica decimal do caractere de valor mais alto na tabela ASCII que aparece na requisição. Esse valor pode representar o uso de um caractere incomum.

Portanto, os dados que chegam efetivamente ao modelo proposto passam por duas etapas de pré-processamento. Primeiramente, pelo descarte de informações que não serão utilizadas da requisição. E, posteriormente, pelo processo de extração de características mais importantes para a detecção de anomalias. Estas características dão origem ao vetor numérico que alimenta as redes neurais do modelo, conforme apresentado pela figura 12.

Figura 12 – Exemplo de extração de recursos numéricos de uma requisição HTTP



Fonte: Próprio autor

4.3 ESTÁGIO 1: FILTRO DE BLOOM.

Há uma necessidade cada vez maior de se construir modelos eficientes, especialmente aqueles que manipulam uma grande quantidade de dados como no caso dos sistemas de detecção de ataques Web. A implementação do modelo apresentado neste trabalho se utiliza da estratégia de redução da dimensionalidade dos dados com o objetivo de torná-lo mais eficiente.

Os métodos de redução de dimensionalidade tentam mapear dados de alta dimensão para dimensões menores, mantendo a estrutura interna dos dados (JUVO-NEN; SIPOLA; HÄMÄLÄINEN, 2015). Espera-se que a aplicação de um etapa prévia de filtragem possa reduzir o fluxo de dados enviado ao detector o que resultaria na diminuição do tempo de detecção sem afetar o desempenho preditivo do modelo.

A seleção de instâncias na proposta deste trabalho é atribuída a filtragem proposta pelo Bloom. Ao preencher o filtro com amostras de requisições legítimas, previamente conhecidas, faz-se a seleção de instâncias benignas que não precisam ser enviadas ao classificador baseado em rede neural. Este processo é fundamental para reduzir o tempo de detecção, uma vez que, o tempo de verificação no Filtro de Bloom é inferior ao processo de classificação da rede neural.

O Filtro de Bloom (BLOOM, 1970) é uma estrutura de dados simples e com espaço eficiente para representar um conjunto e que oferece suporte a consultas. Para implementação desta ferramenta na proposta deste trabalho alguns fatores, como a dinâmica de atuação do filtro e eficiência, são consideradas. Nas sub-seções seguintes a atuação do filtro é apresentado sob estes aspectos.

4.3.1 Dinâmica do Filtro

Para estabelecer a dinâmica de atuação do filtro no modelo proposto por este trabalho analisou-se a sua aplicação em outros trabalhos de pesquisa. Zhou et al. (2014, p. 41) utilizaram o filtro para comportar uma extensa lista de endereços IP de origem mal-intencionados em um sistema de defesa contra ataques DDoS na camada de aplicação. No trabalho de Feng, Li e Chana (2017, p. 6) o filtro recebeu assinaturas de comportamento normal de pacotes de redes. Nos dois casos, assim como na proposta deste trabalho, o filtro precede o classificador.

A diferença destas duas abordagens está relacionada ao que representam os dados que foram usados para preencher o filtro. No primeiro caso (ZHOU et al., 2014), os dados representavam comportamento malicioso e no segundo (FENG; LI; CHANA, 2017) o comportamento normal da rede. A escolha dos dados que vão para o filtro é importante devido à forma como ele atua diante de uma operação de consulta. Ao verificar se um elemento faz parte do conjunto o filtro indicará que ele pode fazer parte do conjunto ou, que ele definitivamente não faz parte do conjunto. Uma aplicação Web contém um número finito de páginas (GROVE, 2009) portanto, as requisições HTTP que representam um comportamento normal também é finito. Sendo assim, no modelo proposto neste trabalho o Filtro de Bloom é utilizado para representar as requisições livres de ataques presentes no conjunto de treino.

Tão pertinente quanto o conteúdo do filtro é saber o que fazer após a consulta. Se uma requisição verificada no Bloom for indicada como não pertencente ao conjunto isso é um fato definitivo. Portanto, uma requisição desconhecida pelo filtro não representa, necessariamente, uma anomalia. O filtro apenas certifica que ela não faz parte do conjunto de requisições normais previamente conhecidas, por isso, ela deve ser encaminhada ao classificador. Ao contrário disso, quando o filtro reconhece uma requisição ela pode ser, de fato, um URL benigna e o modelo a reconhece como um não-ataque.

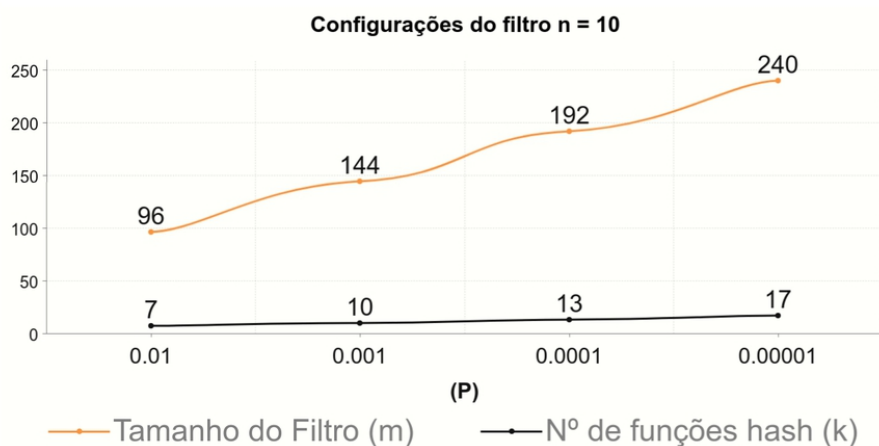
Um retorno positivo do filtro, pode ser sinônimo de uma colisão. A colisão ocorre quando as funções *hash* do elemento consultado apontam para posições do filtro que já estavam preenchidas por outro elemento. Para a dinâmica do modelo proposto neste trabalho, uma colisão no filtro pode representar um falso negativo. A relação 'quantidade de elementos' *contra* 'tamanho do filtro' é que vai definir a probabilidade de colisões. A configuração para evitar colisões é abordada na sub-seção seguinte.

4.3.2 Eficiência do Filtro

Para representar o conjunto de requisições normais o filtro será definido com tamanho m e cada elemento receberá um número k de funções *hash* independentes, conforme Equação (3.1), onde P é a probabilidade de colisões desejada.

A simples redução do valor de P na definição do tamanho do filtro (m) não representa, necessariamente, maior eficiência do filtro. Conforme se reduz o valor de P , haverá aumento no número de funções *hash* (k). A figura 13 ilustra como um filtro a ser configurado com 10 elementos cresce, acompanhado pela número de funções hash, conforme se reduz o valor da probabilidade de colisões.

Figura 13 – Redução do valor de P aumenta o tamanho do filtro e a quantidade de funções hashes.



Considerando que um dos objetivos da proposta deste trabalho passa pela construção de um modelo que possa detectar anomalias com tempo de detecção reduzido buscou-se a implementação mais eficiente para o Filtro de Bloom. Primeiramente, o valor de P foi fixado em 0.01 para evitar o aumento de k . Também optou-se pelo uso da função *Murmur Hash* por se tratar de uma função mais leve e com baixo número de colisões (PARTHASARATHY; KUNDUR, 2012).

Antes, porém, de se efetuar os cálculos que definem a configuração do filtro, as requisições HTTP usadas para preencher o Filtro de Bloom também passam por uma verificação para remover as repetições. A presença de requisições repetidas afeta o

número de elementos (n) que, conseqüentemente, influenciará nos valores de m e k . Isso representaria um aumento desnecessário no tamanho do Filtro.

Portanto, a implementação do primeiro estágio do modelo de detecção de anomalias deste trabalho é alicerçada em uma ferramenta de filtragem leve e eficiente. Ela possui uma atuação dinâmica que vai permitir reduzir as enviadas à rede neural recorrente. Na seção seguinte apresenta-se a dinâmica de atuação da rede neural recorrente dentro do modelo proposto neste trabalho.

4.4 ESTÁGIO 2: REDE NEURAL RECORRENTE

Para o modelo proposto neste trabalho a rede neural recorrente fica encarregada de classificar aquelas requisições que não foram filtradas pelo Filtro de Bloom. Antes, porém, a rede neural precisa ser treinada com amostras rotuladas que indiquem se a requisição é normal ou anômala e, caso seja maliciosa, indique qual tipo de ataque. Nas próximas sub-seções são explanados os processos de treino e classificação no segundo estágio do modelo.

4.4.1 Treinamento da rede neural

As técnicas baseadas em aprendizado de máquina seguem uma sequência de pré-processamento, aprendizado e avaliação dos dados. A fase de pré-processamento (4.2) transforma os dados para um formato que possa ser usado como entrada para o processo de aprendizagem (ZHOU et al., 2017). A seleção de recursos, conforme apresentado na seção 4.2 compreende esta fase, transformando URLs em vetores numéricos com nove valores representando as características da requisição.

No aprendizado de máquina a fase de aprendizagem pode ser realizada com dados em lote, com todos os dados de treino disponíveis ou *online*, quando o modelo atualiza com base em cada nova entrada (ZHOU et al., 2017). Para este trabalho a rede neural será treinada com dados em lote pré-processados para a extração de recursos. Os dados de treino contém amostras de requisições normais e maliciosas, todos rotulados.

A implementação da rede neural contempla, basicamente uma camada de incorporação, uma camada oculta com célula recorrente e a camada densa de saída com função de ativação '*sigmoid*' por se tratar de uma classificação binária. Redes neurais podem ter múltiplas camadas ocultas, porém, a adição de camadas implica em maior complexidade computacional. Por isso, limitou-se a implementação a apenas uma camada oculta.

A definição dos hiperparâmetros para a rede neural recorrente do modelo é feita através de testes com diversas combinações de parâmetros sendo selecionada aquela com maior acurácia. O modelo da célula recorrente também é uma variável do modelo.

Neste trabalho, optou-se por avaliar três categorias de rede neurais recorrentes no segundo estágio, sendo, LSTM, GRU e BI-LSTM.

4.4.2 Classificação

A tarefa de classificação é desempenhada pela rede neural no segundo estágio do modelo proposto neste trabalho. Diferentemente do filtro de Bloom que verifica os URLs extraídas diretamente do cabeçalho da requisição HTTP, o classificador baseado em rede neural recorrente precisa extrair recursos numéricos da requisição antes de realizar a classificação.

É importante salientar que o módulo de extração de recursos atua somente quando um URL não é encontrada no conjunto de requisições normais do filtro de Bloom. Essa dinâmica torna o modelo mais eficiente ao manipular apenas as requisições que vão ao estágio 2. Nesses casos, o módulo extrai nove características do URL (4.2) e a transforma em um vetor numérico. Só então é possível fazer a classificação da requisição HTTP.

A extração de características no momento da detecção simula o ambiente de uma aplicação real e considera esta tarefa para o cálculo de tempo médio de detecção.

4.5 CONSIDERAÇÕES DO CAPÍTULO

Toda a construção do modelo proposto neste trabalho tem elementos que visam obter maior eficiência para o filtro e para a rede neural. No entanto, o desempenho estará estritamente ligado ao volume de dados de treino. Isto porque a implementação de um modelo em dois estágios cria um passo a mais em relação à atuação de um classificador de maneira isolada. Ainda assim, um razoável volume de dados filtrado na primeira fase deve compensar a existência destes passos no algoritmo de detecção.

5 FILTRO DE BLOOM COMO FERRAMENTA DE APOIO AO DETECTOR

Os algoritmos tradicionais de aprendizado de máquina enfrentam desafios críticos de escalabilidade para grandes volumes de dados (ZHOU et al., 2017). Técnicas de aprendizado profundo demonstram bom desempenho em tarefas de reconhecimento e classificação e têm sido preferidas para detecção de anomalias em muitos trabalhos (ONEY; PEKER, 2018). No entanto, o aprendizado profundo possui alto custo computacional e dificuldades de escalabilidade (AL-JARRAH et al., 2015).

Neste capítulo, avalia-se a aplicabilidade do Filtro de Bloom como primeiro estágio do modelo BLOOM-RNN (vide capítulo 4). A função da etapa de filtragem é reduzir a quantidade de requisições que são enviadas ao detector. O filtro é preenchido com requisições normais do conjunto de treinamento. O tempo de verificação de uma requisição no filtro é inferior ao tempo de detecção no detector. Portanto, é demonstrado neste capítulo que a filtragem proporciona uma redução no tempo médio de detecção por requisição sem degradar a qualidade da detecção.

Para uma avaliação mais abrangente da eficiência do filtro foram utilizados, além da rede neural recorrente, mais seis classificadores baseados em aprendizado de máquina. Os classificadores escolhidos foram aplicados em abordagens de detecção de ataques Web de trabalhos recentes (GIMÉNEZ et al., 2015; SAHIN; SOGUKPINAR, 2017; ALTHUBITI; YUAN; ESTERLINE, 2017; BOCHEM; ZHANG; HOGREFE, 2017). São eles: *Decision Tree* (DT), *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVM), *Multilayer Perceptron* (MLP), *Random Forest* (RF), *Artificial Neural Network* (ANN) e a *Recurrent Neural Network* (RNN).

O capítulo está organizado da seguinte forma: A Seção 5.1 detalha de que forma os experimentos estão organizados, apresenta o conjunto de dados utilizado nos testes e as métricas. Em seguida, a Seção 5.2 apresenta detalhes sobre os classificadores utilizados nos experimentos e a configuração do Filtro de Bloom padrão e expandido. Por fim, são discutidos os resultados dos experimentos na Seção 5.3 e as considerações do capítulo na Seção 5.4.

5.1 PLANEJAMENTO DOS EXPERIMENTOS

Para a realização dos experimentos deste capítulo, considera-se exatamente a proposta do modelo exposta na seção 4.3 que especifica a atuação do Filtro de Bloom. Os experimentos foram divididos em três partes para que fosse possível medir os impactos da aplicação do filtro de Bloom ao detector.

- 1) Primeiramente, fez-se necessário o treinamento e teste das técnicas de aprendizado de máquina individualmente para que houvessem valores de referência que permitissem a comparação do modelo após aplicação do filtro.

- 2) Posteriormente, aplicou-se o filtro de Bloom combinado a cada uma das técnicas escolhidas para classificação. Neste experimento, o filtro de Bloom é configurado para ter um tamanho e número de funções hash que permitam ao modelo obter uma taxa de 1% de colisão.
- 3) Na terceira parte dos experimentos procurou-se explorar a configuração do Filtro de Bloom (otimizações) para reduzir as colisões sem afetar a precisão preditiva e o tempo de detecção.

As configurações do computador utilizado nestes experimentos foi: Processador Quad core Intel Core i7-4790, cache de 8192 KB, 7882 MB de memória RAM e aceleração GPU GeForce GTX 745, com sistema operacional Ubuntu 18.04 64 bits.

5.1.1 Conjunto de Dados CSIC 2010

No experimento deste capítulo foi utilizado o HTTP DATASET CSIC 2010 (CSIC, 2010). Este conjunto de dados aparece em onze de quinze trabalhos citados na seção de trabalhos relacionados (vide seção 2.4). O conjunto de dados do CSIC versão 2010 possui 35992 requisições normais e 24664 anômalas. O conjunto contempla ataques estáticos e dinâmicos, incluindo ataques modernos da Web, especialmente os ataques de injeção de código.

Para avaliar a utilização do Filtro de Bloom com as técnicas de aprendizado de máquina foram utilizados os dados normais e anômalos do conjunto CSIC. Eles foram mesclados em dois conjuntos de teste e de treino. O conjunto de treino, com 42459 amostras (70% da base), foi utilizado para preencher o filtro de Bloom e treinar cada uma das 7 técnicas de aprendizado de máquina. Já o conjunto de testes, com 18197 amostras (30% da base), foi utilizado para avaliar o desempenho do modelo.

5.1.2 Métricas

Para avaliar os experimentos propostos neste capítulo calculou-se a acurácia (ACC), a precisão (P) e a taxa de detecção (DR) com base em uma matriz de confusão, conforme figura 14, sendo: TP (True Positive), TN (True Negative), FP (False Positive) e FN (Falso Negative). As métricas apresentadas na Tabela 3 foram incluídas para análise do desempenho preditivo. Apesar de o desempenho não ser o foco dos experimentos neste capítulo, a presença do filtro na estrutura de detecção não deve influenciar negativamente no desempenho.

Figura 14 – Matriz de confusão

	PREDICT		
REAL		NORMAL	ANOMALY
NORMAL		TRUE NEGATIVE (TN)	FALSE POSITIVE (FP)
ANOMALY		FALSE NEGATIVE (FN)	TRUE POSITIVE (TP)

Além destas métricas de detecção, também foi analisado o Tempo Médio de Detecção (AVG). O tempo médio é calculado através da média aritmética dos tempos registrados durante a verificação de cada requisição no filtro e no classificador. Na próxima seção são apresentados os detalhes da implementação dos detectores utilizados nos experimentos deste capítulo.

Tabela 3 – Métricas de desempenho preditivo

Métrica	<i>Accuracy</i>	<i>Precision</i>	<i>Detection Rate</i>
Abreviação	ACC	P	DR
Equação	$\frac{TP+TN}{TP+TN+FP+FN}$	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$

A acurácia

5.2 IMPLEMENTAÇÃO E CONFIGURAÇÃO DOS DETECTORES

Para a implementação dos classificadores foi utilizada a linguagem Python com apropriação dos recursos das bibliotecas *Scikit-learn* e *Keras*. *Scikit-learn* é uma biblioteca de aprendizado de máquina que fornece implementações eficientes de algoritmos de última geração (ABRAHAM et al., 2014). O *Keras* é uma API para redes neurais de alto nível, escrita em Python e capaz de executar sobre o *Tensor Flow*, *CNTK* e *Theano* (CHOLLET, 2015).

Os modelos implementados no *Scikit-learn* recebem dados de entrada na forma de matrizes bidimensionais de amostras de tamanho *vs* recursos (ABRAHAM et al., 2014). Assim, foi utilizado o módulo de extração de recursos do modelo (Figura 10) para transformar as requisições em vetores numéricos que representam as características das requisições HTTP (vide seção 4.2). A partir do *Scikit-learn* foram implementados os classificadores:

- Support Vector Machine - SVM
- Multilayer Perceptron - MLP
- K-Nearest Neighbor - KNN

- Random Forest - RF
- Decision Tree - DT

A partir do Keras foram implementados os detectores baseados em redes neurais. Uma rede neural artificial com apenas uma camada de entrada com função de ativação *softsign* e uma camada de saída com função *sigmoid* densamente conectadas. A rede neural recorrente recebeu uma camada de incorporação, comumente utilizada para a classificação de textos, uma camada *Long Short-Term Memory* (LSTM), que possui células de memória que ajudam a resolver o problema de dependência de longo prazo (ALTHUBITI; YUAN; ESTERLINE, 2017), uma camada de *dropout* com 0.2 e uma camada de saída com função de ativação *sigmoid*.

- Artificial Neural Network - ANN
- Recurrent Neural Network - RNN

Todos os classificadores receberam configurações básicas em sua implementação e não houve exploração de configuração dos parâmetros, dado que, alcançar altos níveis de precisão não é o objetivo deste capítulo. Isto será explorada posteriormente para a rede neural recorrente.

5.2.1 Detecção com Filtro de Bloom (D-BLOOM)

Do conjunto de treino foram utilizadas apenas os URLs rotulados como “normal” para preencher o Filtro de Bloom. Neste processo também são removidos os URLs repetidos restando 4399 URLs normais para preencher o filtro. Tendo conhecimento dessa quantidade de elementos foram, então, calculados o tamanho do filtro (equação 3.1) e o número de funções *hash* (Equação 3.2) ideais para se obter uma taxa de colisão de 1%. O filtro deste experimento tem um tamanho de 42165 posições (m) com 7 funções de hash (k).

No processo de detecção, cada amostra é verificada no Filtro de Bloom e somente aquelas ausentes no filtro são encaminhadas ao detector que faz a classificação do URL como normal ou anômalo. Amostras presentes no filtro, incluindo as colisões, foram classificadas automaticamente como normais por ter um elemento semelhante no filtro. A aplicação do filtro (experimento D-BLOOM) representou uma melhora na acurácia e precisão, bem como, no tempo de detecção.

5.2.2 Detecção com Filtro de Bloom Expandido (D-BLOOM+)

Mesmo configurado para obter uma taxa de colisão de 1%, o uso do filtro implicou em aumento de falsos negativos decorrente das colisões, o que já era esperado

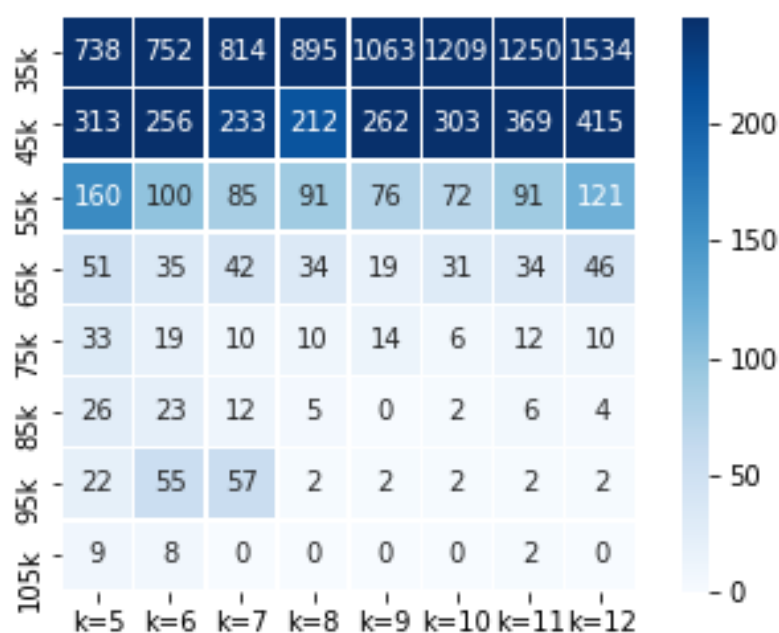
(detalhamento dos resultados serão apresentados na seção 5.3). Dado que o filtro se mostrou eficaz, um novo experimento foi realizado com um filtro de Bloom expandido (D-BLOOM+). Neste experimento foram exploradas possibilidades de configuração do filtro de Bloom para minimizar os falsos negativos gerados pela presença do filtro. A seguir é detalhado como este experimento foi implementado e configurado.

5.2.2.1 Otimização das configurações do Filtro Bloom

Para este experimento de otimização do filtro foram utilizadas 4399 requisições normais para preencher o filtro e 24664 requisições anômalas para consulta. Neste teste, os elementos consultados não devem aparecer no filtro, uma vez que, os dois conjuntos apresentam dados completamente distintos.

Com base na configuração obtida a partir das equações 3.1 e 3.2 apresentadas na seção 3.3, sendo $P=0,01$, o filtro teria tamanho $m=42165$ e $k=7$ para a quantidade de elementos $n=4399$. Na figura 15 observa-se a variação de combinações para m (35k, 45k, 55k, 65k, 75k, 85k, 95k, 105k) e k (5, 6, 7, 8, 9, 10, 11, 12). Os valores na matriz representam o número de colisões para cada combinação.

Figura 15 – Quantidade de colisões no filtro de acordo com a configuração



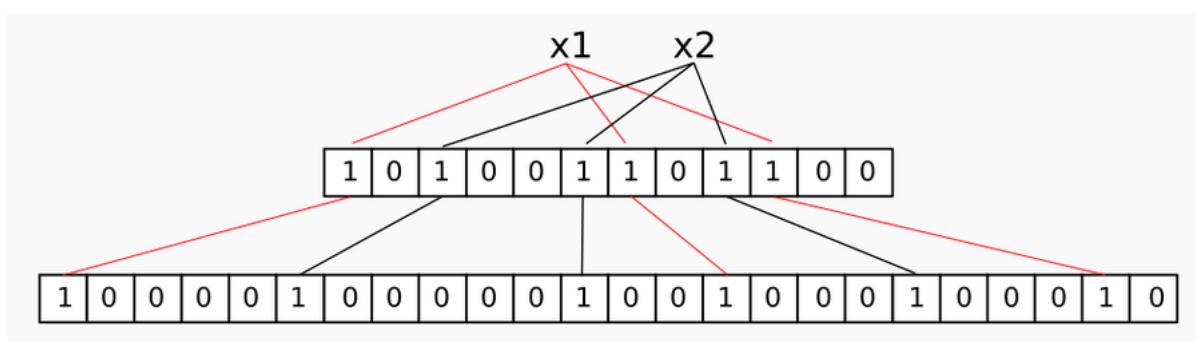
Observe que optar por uma simples redução do valor de P implicaria no aumento do tamanho do filtro e do número de funções *hash*. Aumentar a quantidade de funções *hash* poderia implicar em aumento no tempo de detecção. No entanto, observando a figura 15 é possível perceber que ao dobrar o tamanho do filtro o número de colisões reduz significativamente, sem necessidade de aumentar as funções *hash*. O filtro com tamanho 45k e 7 funções *hash* apresentou 233 colisões. Enquanto, com 95k e 7 funções *hash* o filtro apresentou 57 colisões.

Com esta análise considerou-se a configuração do modelo de detecção com um filtro de Bloom com o dobro de tamanho do aplicado ao experimento anterior conforme a equação 5.1.

$$m = - \left(\frac{n \ln(P)}{(\ln 2)^2} \right) * 2 \tag{5.1}$$

A Figura 16 ilustra o que acontece com a expansão. Um filtro de tamanho dobrado para a mesma quantidade de elementos e com mesmo número de funções hash. Com o filtro expandido a possibilidade de colisões é muito menor.

Figura 16 – Filtro de Bloom expandido com tamanho dobrado em relação à configuração padrão



Fonte: Próprio autor

5.2.2.2 Detecção com o Filtro Expandido

A comparação da configuração do D-BLOOM com o D-BLOOM+ possibilita compreender melhor as potencialidades do modelo de detecção com Filtro de Bloom. No experimento D-BLOOM+ o filtro foi configurado para um tamanho de 84330 posições (m), o dobro do tamanho original, e 7 funções de hash (k). O filtro foi preenchido com as URLs do tipo “normal” (4399 URLs) presentes no conjunto de treino e os testes de detecção foram realizados com o conjunto de testes. A tabela 4 apresenta a configuração do filtro expandido em comparação ao modelo anterior. Salienta-se que a única diferença é de 42kbits de tamanho.

Tabela 4 – Configurações dos Filtros Bloom padrão e expandido

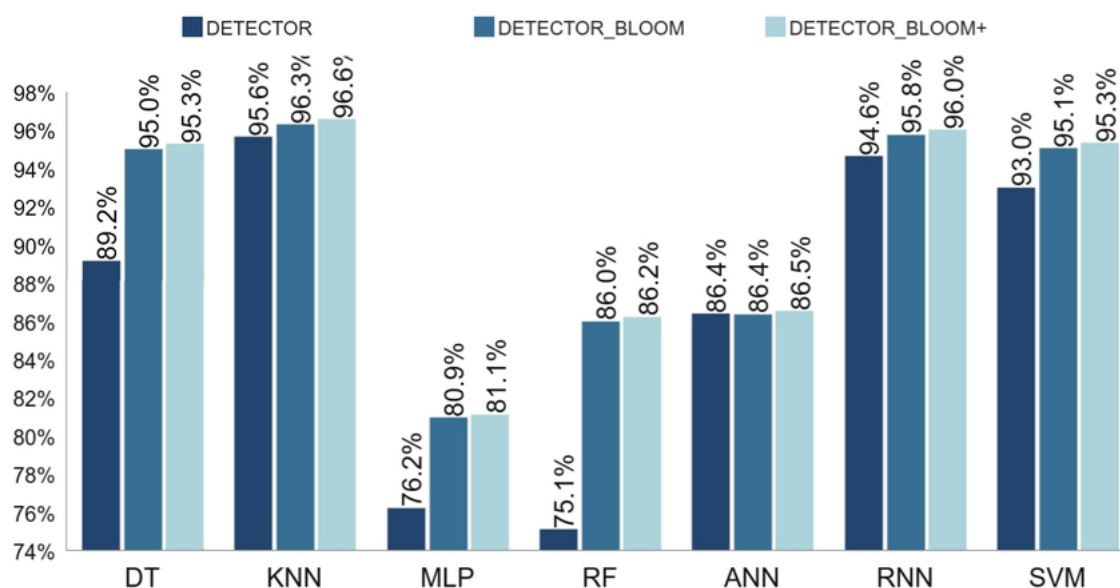
Modelo	Elementos (n)	Tamanho (m)	Hash (k)
D-BLOOM	4399	42165	7
D-BLOOM+	4399	84330	7

5.3 RESULTADOS E DISCUSSÃO

Neste capítulo, o principal objetivo com os experimentos foi analisar os impactos da aplicação do Filtro de Bloom, como primeiro estágio na estrutura proposta no modelo deste trabalho. O propósito foi reduzir a quantidade de elementos (requisições) enviados ao detector, diminuindo assim o tempo de detecção da estrutura sem afetar negativamente as métricas de Acurácia, Precisão e Taxa de Detecção. Desta forma, analisou-se cada uma das técnicas de aprendizado de máquina individualmente, combinadas ao Filtro de Bloom (D-BLOOM) e ao Filtro de Bloom Expandido (D-BLOOM+).

A Figura 17 apresenta os resultados referentes a medida de acurácia (ACC). Para cada um dos sete algoritmos são apresentados os resultados sem a presença do Bloom (DETECTOR), com a presença do Bloom (DETECTOR_BLOOM) e com o filtro expandido (DETECTOR_BLOOM+). Na acurácia todos os resultados da matriz de confusão são considerados.

Figura 17 – Gráfico com resultados da métrica de Acurácia (ACC)



Fonte: Próprio autor

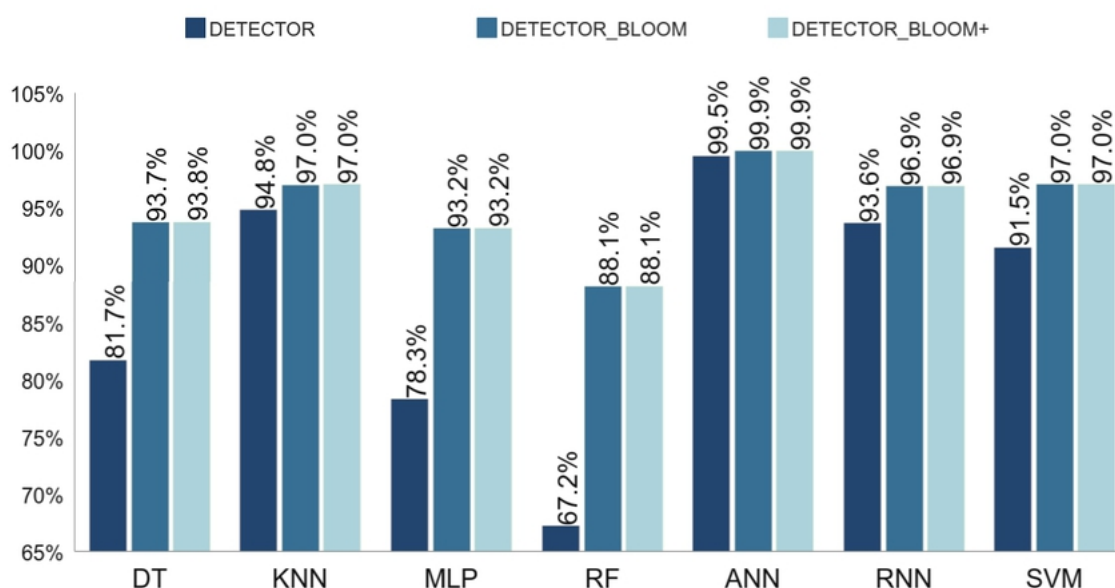
Todos os classificadores tiveram aumento da acurácia com a presença do filtro de Bloom e mais um pequeno acréscimo com o filtro expandido, à exceção da rede neural artificial (ANN) que manteve o índice com filtro, mas que ainda assim, teve pequeno acréscimo com filtro expandido.

Os classificadores que apresentaram baixo desempenho com medidas abaixo dos 90% foram os que tiveram maior ganho com a presença do filtro. Atuando isoladamente o *Decision Tree* (DT) foi de 89.2% para 95% com o filtro padrão e 95.3% com o expandido. MLP e RF saltaram da casa dos 70% para níveis acima dos 80%. A rede

neural recorrente (RNN) que obteve 94.6% de acurácia teve seu índice elevado para 95.8% com o filtro de Bloom e 96% com o filtro de Bloom Expandido.

Os gráficos da Figura 18 revelam que a presença do Filtro de Bloom também representou aumento na precisão (P) de todos os classificadores testados. A métrica de precisão faz um balanço entre a presença de verdadeiros positivos e falsos positivos durante a detecção. A presença do filtro reduz a probabilidade de falsos positivos por descartar no primeiro estágio do modelo as requisições reconhecidamente legítimas.

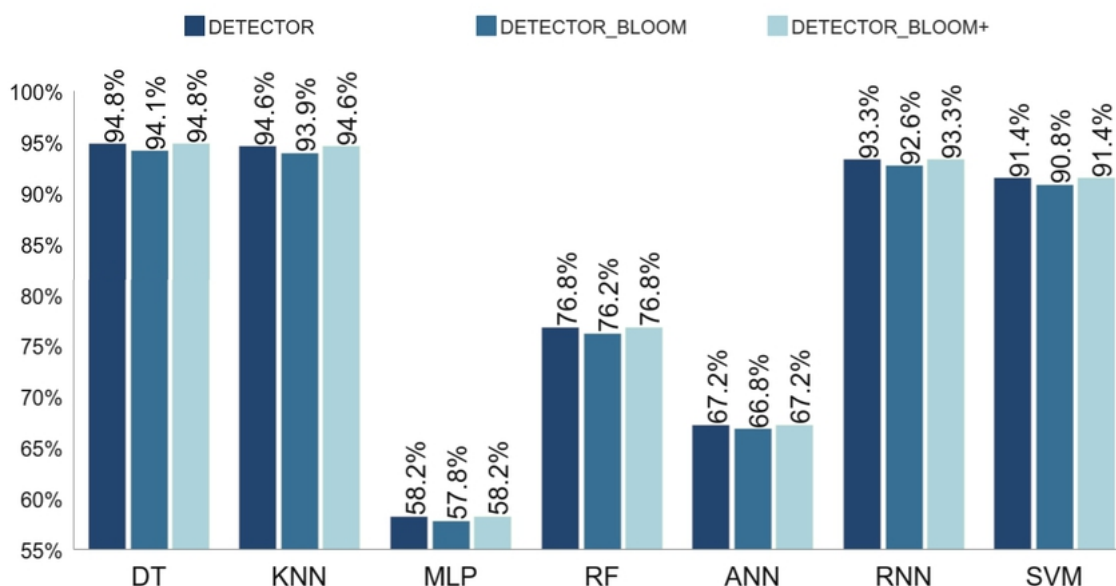
Figura 18 – Gráfico com resultados da métrica de Precisão (P)



Para esta medida é possível observar que a presença do filtro de Bloom expandido tem pouca influência. Na comparação com o modelo DETECTOR_BLOOM os valores são iguais ou muito próximos. Isto ocorre porque a expansão do filtro tem a função de reduzir colisões que no contexto do modelo combinado representam falsos negativos na matriz de confusão. Falsos negativos não são considerados na métrica de precisão.

A Taxa de Detecção (DR), no entanto, é medida com base em verdadeiro positivo e falso negativo. Na Figura 19 é possível observar que a medida apresentou leve queda em todos os classificadores quando combinados com filtro padrão (D-BLOOM). Estes dados corroboram a importância do Filtro de Bloom Expandido. Sua versão convencional com configuração padrão gerou muitas colisões e afetou as taxas de detecção em todos os classificadores. Porém, estas colisões foram sanadas na versão expandida (D-BLOOM+) permitindo manter as taxas de detecção dos classificadores.

Figura 19 – Gráfico com resultados da métrica de Taxa de Detecção (DR)



Fonte: Próprio autor

As métricas preditivas demonstram que a aplicação do Filtro de Bloom, quando corretamente configurado, além de auxiliar na redução de dados de entrada para os classificadores, ainda promove um pequeno aumento das taxas de precisão e acurácia. Há incremento nas métricas preditivas com a presença do Filtro mesmo utilizando a configuração básica fornecida pelas ferramentas *Scikit-learn* e *Keras* para os classificadores.

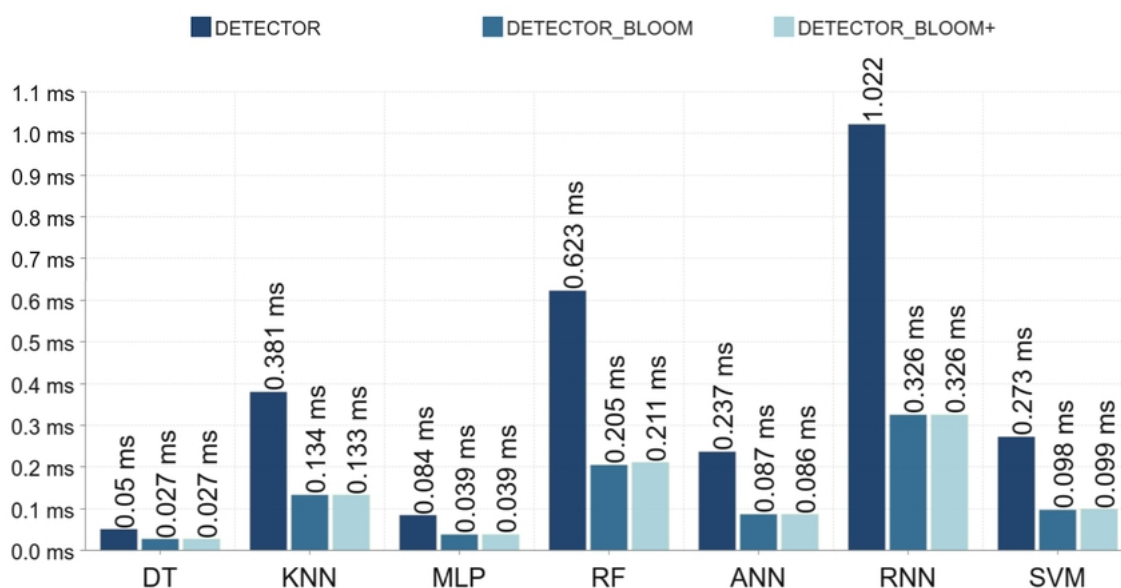
O tempo de detecção é a principal medida adotada nos experimentos para analisar a eficiência do modelo BLOOM-RNN. O ganho em eficiência neste cenário representa a redução na quantidade de requisições enviadas ao detector baseado em *machine learning*, consequentemente reduzindo o tempo médio de detecção. O tempo que o servidor leva para fazer uma classificação é superior ao tempo de verificação no filtro que é baseado em *hash*.

Com 18197 amostras no conjunto de teste o Filtro de Bloom padrão foi capaz de filtrar 9946 requisições normais, o que representa 54,65% das verificações. Com o Filtro de Bloom Expandido a quantidade de requisições filtradas foi de 9891 representando um percentual de 54,35%. A redução desse valor é resultado da eliminação das colisões após a configuração do filtro expandido. Saliencia-se aqui que estes valores para a etapa de filtragem são idênticos para todos os classificadores, pois, os conjuntos de treino e teste foram os mesmos e não há variação na filtragem por se tratar de verificação de *hash*.

Sendo assim, a presença do filtro provocou uma redução no tempo médio que cada requisição leva para passar pela estrutura de detecção. É possível observar na

Figura 20 que essa redução no tempo ultrapassa 50% para todas os classificadores testados. Os resultados com Filtro de Bloom padrão e expandido são relativamente idênticos. Como mencionado anteriormente, a diferença na quantidade de requisições filtradas do filtro padrão para o expandido foi de apenas 55 requisições, porém, foi crucial para evitar que a presença do filtro fosse negativa na métrica de taxa de detecção.

Figura 20 – Gráfico com resultados da métrica de Tempo Médio de Detecção (AVG)



Fonte: Próprio autor

A rede neural recorrente (RNN) testada, que apresenta maior tempo médio com 1.022ms, apresentou uma ótima redução para 0.326ms com Bloom padrão e expandido. O filtro também se mostrou eficiente mesmo para as técnicas que haviam apresentado um tempo médio de detecção baixo sem a presença do filtro. O *Decision Tree* (DT) reduziu seu tempo médio de detecção de 0.05ms para 0.027ms com o filtro. Mesma situação observada para o *Multilayer Perceptron* (MLP) que reduziu o tempo médio de 0.084ms para 0.039ms com filtro.

Estes resultados indicam que o filtro tem potencial para apoiar classificadores baseados em aprendizado de máquina, especialmente aqueles de custo computacional elevado.

5.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram testados sete classificadores baseados em aprendizado de máquina comumente utilizados na detecção de anomalias para ataques em aplicações da Web. A presença do Filtro de Bloom como etapa prévia de classificação conseguiu

reduzir a quantidade de requisições enviadas aos classificadores. O Filtro de Bloom padrão foi capaz de filtrar 54,65% das requisições e 54,35% com o Filtro de Bloom Expandido. Com esta atuação o filtro demonstrou ser eficiente na redução do tempo médio de detecção com todas as técnicas testadas nos experimentos.

Com foco nos potenciais do Filtro de Bloom, neste capítulo não houve exploração dos parâmetros de configuração dos classificadores. Mesmo assim, a presença do Filtro de Bloom representou melhoria nas taxas de precisão e acurácia em todos os classificadores. A exploração nas configurações do filtro que deram origem ao Bloom Expandido foram fundamentais para eliminar as colisões que ocorreram com o uso do Filtro de Bloom convencional e manter estável as taxas de detecção em todos os classificadores.

Nesta fase da pesquisa foi possível tornar detectores baseados em *machine learning* mais eficientes (tempo médio de detecção) e eficazes (predições) com a presença do Filtro de Bloom. Os resultados parciais desenvolvidos até este ponto da pesquisa foram publicados em artigo no XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2019).

6 ANÁLISE DE REDES RECORRENTES APLICADAS AO MODELO DE DETECÇÃO

As Redes Neurais Recorrentes vêm sendo bastante utilizadas na detecção de anomalias em ataques contra aplicações da Web. A maior parte dos trabalhos recentes que utilizou redes recorrentes em detecção de ataques Web utilizou o modelo *Long Short Term Memory* (LSTM) (BOCHEM; ZHANG; HOGREFE, 2017; ALTHUBITI et al., 2018; KIM; CHO, 2018; WANG; ZHOU; CHEN, 2018). O modelo *Gated Recurrent Unit* (GRU) é uma versão mais simples da rede LSTM e também figura nos trabalhos de pesquisa (ZHAO et al., 2018; LIANG; ZHAO; YE, 2017a). Há ainda as pesquisas que se utilizaram da versão *Bidirecional* das redes LSTM (HAO; LONG; YANG, 2019; YU et al., 2018).

Neste capítulo explora-se estas três categorias de redes neurais recorrentes que foram apresentadas na fundamentação teórica (ver seções 3.2.1, 3.2.2 e 3.2.3). Esta é uma importante contribuição deste trabalho, visto que, não foram encontrados trabalhos de pesquisa no domínio de segurança de aplicações da Web que tenham exposto uma análise comparativa entre as três categorias de redes recorrentes aqui utilizadas. É possível que não haja trabalhos comparativos em função de as RNNs não terem apresentado anteriormente competitividade, o que mudou com a demonstração, neste trabalho, de que a inserção do filtro (Bloom) pode reduzir o volume de dados a serem analisados.

Primeiramente, busca-se os hiperparâmetros para que cada rede alcance maior eficácia. Para isso, no entanto, utilizou-se uma ferramenta do *Scikit-learn* (ABRAHAM et al., 2014) chamada *GridSearchCV* para encontrar os hiperparâmetros para a configuração de cada rede. O *GridSearchCV* é uma tarefa de testes de combinação de vários valores para a rede neural na busca pela melhor configuração (PEDREGOSA et al., 2011). Devidamente configuradas com os hiperparâmetros, as redes neurais recorrentes são aplicadas ao segundo estágio do modelo de detecção proposto na seção 4.4 e combinadas ao Filtro de Bloom Expandido como apresentado na seção 5.2.2.2.

Dois cenários de experimentação são aplicados. Primeiramente com a presença de grande volume de dados a exemplo do capítulo 5 e tem por objetivo comparar as redes neurais recorrentes entre si. Em seguida, o modelo é avaliado com 6 categorias de ataques presentes em uma versão do conjunto de dados CSIC do ano de 2012 (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2012) que apresenta ataques categorizados (o detalhamento deste conjunto de dados é apresentado na Seção 6.3). Este experimento visa a avaliação do modelo BLOOM-RNN diante de ataques específicos com cada uma das redes recorrentes.

A seguir a Seção 6.1 apresenta a organização dos experimentos realizados com as redes neurais recorrentes e detalha o processo de busca pelos hiperparâmetros. Ainda nesta seção é apresentado o conjunto de dados CSIC 2012 também utilizado

nos experimentos deste capítulo. A Seção 6.2 faz uma análise comparativa das redes neurais recorrentes LSTM, BI-LSTM e GRU combinadas com o Filtro de Bloom expandido e devidamente configuradas com seus hiperparâmetros. A Seção 6.3 avalia o desempenho individual de cada rede neural diante de seis categorias de ataques presentes no conjunto de dados CSIC 2012. Ao final, a Seção 6.4 faz uma síntese com as considerações sobre o que foi desenvolvido no capítulo.

6.1 ORGANIZAÇÃO DOS EXPERIMENTOS

Para validar a proposta do modelo de detecção que contempla o Filtro de Bloom Expandido com Rede Neural Recorrente dois experimentos são realizados. Nos experimentos foram explorados os três modelos de redes neurais recorrentes mencionados no referencial teórico (vide Seção 3.2).

O primeiro experimento é uma continuação do capítulo 5, desta vez, com foco no detector baseado em rede neural. As redes LSTM, BI-LSTM e GRU são aplicadas ao modelo individualmente. A dinâmica do experimento é a mesma proposta na seção 5.1.1 onde as amostras de URLs normais e anômalos são concatenadas e depois divididas em conjuntos de treino e teste com 70% e 30% das amostras, respectivamente.

O segundo experimento testou a eficácia do modelo aplicando individualmente seis categorias de ataques presentes no conjunto de dados CSIC 2012 (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2012). Este conjunto é uma versão atualizada do conjunto de dados CSIC 2010 utilizado nos experimentos do Capítulo 5.

6.1.1 Seleção de hiperparâmetros para as redes neurais recorrentes

A seleção de hiperparâmetros¹ para uma arquitetura de rede neural é fundamental para alcançar o melhor desempenho e há poucos trabalhos que apresentam informações detalhadas para alcançar os valores ideais (REIMERS; GUREVYCH, 2017). Neste trabalho foi utilizada a abordagem em grade denominada *GridSearchCV*. Trata-se de um recurso do *Scikit-learn* que permite encontrar hiperparâmetros para uma rede usando a validação cruzada e distribuindo opcionalmente a computação para vários núcleos (PEDREGOSA et al., 2011).

Apesar de ser uma abordagem que tem sua complexidade e tempo aumentados conforme se eleva o número de possibilidades, o *GridSearchCV* permite definir uma grade com os hiperparâmetros que se pretende experimentar. Considerou-se avaliar a melhor configuração para os seguintes parâmetros: tamanho do Lote (Batch Size), o Número de Épocas no treinamento (Epochs), a Taxa de Esquecimento (Dropout), o

¹ A palavra Hiperparâmetros não existe no português mas é um termo utilizado para se referir aos melhores parâmetros de configuração no campo do aprendizado de máquina.

Otimizador (Optimizer), o número de dimensões de saída (Output Dim) e o número de unidades na célula recorrente (Units).

O resultado do processo de ajuste dos hiperparâmetros das redes neurais pode ser visto na tabela 5 com destaque para os melhores parâmetros encontrados para cada das células recorrentes. Foram testadas individualmente as redes do tipo LSTM, BI-LSTM e GRU. Os itens de configuração escolhidos para testes no *GridSearchCV*, assim como os valores testados, se deram pela observação de diversos valores testados em experimentos preliminares aliados ao conhecimento adquirido na literatura.

Os seis itens testados com três valores cada um resultam em 729 combinações possíveis. Aplica-se ainda a validação cruzada em 3 iterações totalizam 2187 simulações. Nesta etapa de ajuste optou-se por um conjunto reduzido do conjunto de dados com 2800 requisições HTTP para treino e 1400 para testes. Ainda assim, cada execução levou cerca de 20 horas para testar as 2187 simulações.

Tabela 5 – Resultados do GridSearchCV para hiperparâmetros das redes neurais.

Item	Valores Testados	LSTM	BI-LSTM	GRU
Batch Size	10 30 60	10	60	10
Dropout	0 0.1 0.2	0	0.1	0
Epochs	10 30 50	10	50	10
Output Dim	32 64 128	128	64	128
Optimizer	Adam Nadam RMSprop	rmsprop	rmsprop	nadam
Units	30 50 100	100	30	30

Os resultados apontaram que o RMSProp é o otimizador mais indicado para as redes LSTM e BI-LSTM e o Nadam para a rede GRU. Tendo isto definido, foi verificada a melhor taxa de aprendizagem (*Learning Rate*). No entanto, os testes apontaram para os valores padrões definidos pelo Keras. Para o *RMSprop* foi mantido `learning_rate=0.001` e para o otimizador *Nadam* o `learning_rate=0.002`.

6.2 ANÁLISE DAS REDES RECORRENTES NO MODELO BLOOM-RNN

Nesta seção são apresentados os resultados preditivos para o modelo BLOOM-RNN com filtro de Bloom expandido. As redes neurais são configuradas com seus hiperparâmetros apresentados na Seção 6.1.1 são aplicadas individualmente no modelo. Mede-se então, a Acurácia (ACC), a Precisão (P), a Taxa de Detecção (DR) e tempo médio de detecção (AVG). Para corroborar as benesses do Filtro de Bloom aplicam-se estas métricas também em cenário de aplicação da rede neural recorrente sem o Filtro de Bloom.

A tabela 6 apresenta os resultados dos testes comparativos. Nela é possível perceber que o Bidirecional LSTM obteve o melhor índice para as medidas de Acurácia (0.9577) e Precisão (0.9689). Apenas na métrica de Taxa de Detecção a atuação da rede GRU com filtro (BLOOM+GRU) foi superior, mas com superioridade singela (0.9270 contra 0.9266 da BI-LSTM). A combinação entre Filtro de Bloom e Rede Neural Recorrente é sempre superior ao cenário em que as redes atuam sozinhas e isso é verdadeiro para todas as métricas. O que ratifica os experimentos do Capítulo 5 e mostra que a presença do filtro não degrada o modelo, ao contrário disso, há um acréscimo nas medidas quando há filtragem.

Tabela 6 – Comparativo entre índices de predição para as redes neurais recorrentes

Rede Neural	ACC	P	DR	[TN FN] [FP TP]
LSTM	0.9304	0.9181	0.9113	[10139 606] [661 6791]
BLOOM+LSTM	0.9520	0.9649	0.9161	[10497 248] [625 6827]
BI-LSTM	0.9313	0.9177	0.9141	[10134 611] [640 6812]
BLOOM+BI-LSTM	0.9577	0.9689	0.9266	[10523 222] [547 6905]
GRU	0.9333	0.9327	0.9022	[10260 485] [729 6723]
BLOOM+GRU	0.9563	0.9648	0.9270	[10493 252] [544 6908]

Fonte: Dados do experimento

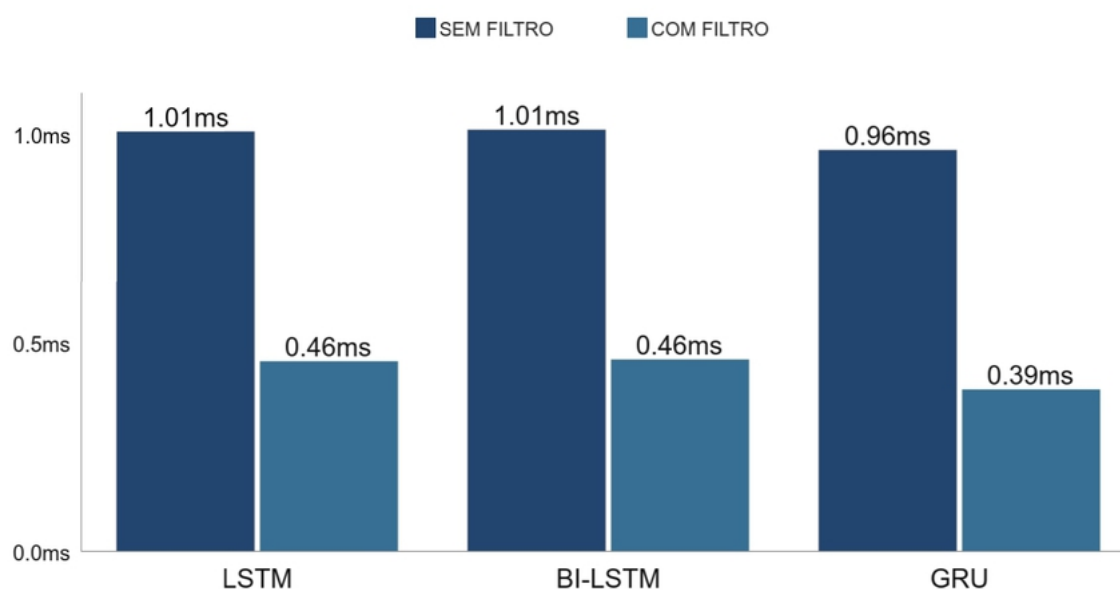
A última coluna presente na tabela 6 apresenta os valores detalhados da detecção conforme a matriz de confusão (apresentada na Figura 14). Por estes valores, percebe-se que o baixo número de Falsos Positivos (FP) é o principal responsável pela boa atuação da BI-LSTM. A menor quantidade de Falsos Negativos (FN) garantiu o maior índice de taxa de detecção.

A LSTM é a rede recorrente mais utilizada em trabalhos de detecção de anomalias no domínio de aplicações Web entre os trabalhos expostos na Seção 2.2. Grande parte destes trabalhos tem buscado alcançar índices preditivos cada vez maiores. No entanto, nos resultados apresentados na Tabela 6 ela tem índices inferiores às concorrentes GRU e LSTM Bidirecional. São diferenças muito pequenas, afinal, as redes possuem o mesmo fundamento em sua arquitetura, mas que são passíveis de consideração quando se trata de medidas preditivas. A superioridade de detecção da

rede LSTM Bidirecional é um resultado esperado uma vez que, esta rede atua como se fossem duas redes LSTM convencionais em sentidos opostos.

A Figura 21 mostra a diferença de tempo médio de detecção entre as redes recorrentes nos cenários com e sem o filtro. Mais uma vez, o tempo médio de detecção inferior com a presença do filtro confirma a melhoria resultante da filtragem na primeira etapa do modelo BLOOM-RNN. Comparando as redes entre si, sob o aspecto de tempo de detecção, percebe-se que esta métrica acompanha a complexidade da estrutura de cada rede neural recorrente. A Bidirecional LSTM tem o maior tempo médio com 0.46ms, seguida da LSTM convencional com 0.44ms. O menor tempo médio de detecção fica com a rede que tem a arquitetura mais simples dentre as três. A GRU obteve tempo médio de detecção de 0.39ms.

Figura 21 – Comparativo de tempo de detecção entre as redes recorrentes em cenários com e sem o Filtro de Bloom



Fonte: Próprio autor

O tempo médio de detecção é uma métrica de difícil comparação com outros trabalhos, pois, depende diretamente da configuração do computador onde são realizados os experimentos. Ainda assim, a redução expressiva do tempo médio de detecção apresentada em todos os cenários com Filtro de Bloom é empolgante.

Para se ter um panorama preciso da avaliação das três redes recorrentes testadas se faz necessário medi-las sob os aspectos de tempo e predição. Esta é uma medida complexa por envolve duas grandezas que se mede em sentidos opostos. Quanto maior a acurácia, melhor o modelo, mas para o tempo médio de detecção, menor é melhor. Adaptou-se a métrica *Adjusted Ratio of Ratios* (ARR) que combina informações sobre a acurácia (ACC) e o tempo (AVG) de algoritmos de aprendizado de

máquina (BRAZDIL; SOARES; COSTA, 2003). Nesta métrica, expressa na Equação 6.1, o parâmetro $AccD$ é arbitrário e pode ser ajustada uma pontuação que considere mais tempo ou mais a acurácia. A configuração $AccD = 0.01$ representa um compromisso mais equilibrado entre os dois critérios (BRAZDIL; SOARES; COSTA, 2003).

$$ARR = \frac{ACC}{1 + AccD * \log(AVG)} \quad (6.1)$$

A tabela 7 faz um *ranking* das redes neurais recorrentes de acordo com a pontuação obtida pela medida ARR. A rede Bidirecional LSTM combinada com Filtro de Bloom tem a maior pontuação. Apesar de ser levemente mais lenta que as concorrentes, a BI-LSTM consegue se manter competitiva graças aos bons índices preditivos. A rede GRU com Filtro de Bloom aparece em seguida. Neste caso, o baixo tempo de detecção fez com que ela superasse a rede LSTM combinada com Filtro de Bloom.

Tabela 7 – Ranking da melhor rede baseado na métrica ARR

Ranking	AccD = 0.01	
	RNN	ARR
1	BLOOM+BI-LSTM	.9544
2	BLOOM+GRU	.9522
3	BLOOM+LSTM	.9486
4	GRU	.9331
5	BI-LSTM	.9313
6	LSTM	.9304

As redes neurais aplicadas sem Filtro de Bloom tiveram índices menores de ARR, o que era esperado. A rede GRU obteve maior equilíbrio, seguida da rede LSTM Bidirecional. A LSTM teve o pior resultado com esta métrica que envolve tempo e acurácia.

6.3 ANÁLISE DE DESEMPENHO DO MODELO POR ATAQUE

Este experimento visa a análise do modelo BLOOM-RNN com diferentes redes neurais recorrentes diante de diferentes categorias de ataques. Neste caso específico, avaliou-se o comportamento do modelo sob o aspecto da precisão preditiva.

No entanto, o conjunto de dados CSIC 2010 (CSIC, 2010) não possui dados de ataque categorizados. Desta forma, optou-se pela adoção de uma versão atualizada deste mesmo conjunto, o CSIS 2012 (TORRANO-GIMENEZ; PEREZ-VILLEGAS; ALVAREZ, 2012). O HTTP DATASET CSIC 2012 é público e foi criado artificialmente com

objetivo de avaliar mecanismos de defesa contra aplicações da Web. Os dados contêm solicitações da web destinadas a uma aplicação de comércio eletrônico. São disponibilizados em arquivos XML e contêm todas as informações da mensagem (HTTP) que é enviada ao servidor Web.

A tabela 8 apresenta as categorias de ataques presentes no conjunto, bem como, as quantidades de cada uma. A presença de ataques do tipo *SQL Injection* é predominante no conjunto com mais de 43 mil amostras. As categorias de ataques *LDAPi* e *Format String* (que aparecem tachadas na tabela) estão presentes no conjunto de dados, mas não foram consideradas para os experimentos desta seção porque possuem quantidade inferior a 100 amostras no conjunto.

Tabela 8 – Tipos e quantidades de dados normais e ataques presentes no conjunto CSIC 2012.

Categorias de ataques	Amostras
Dados normais	8363
Ataques XSS	4818
Ataques SQLi	43013
Ataques Xpath	175
Ataques LDAPi	74
Ataques SSI	451
Ataques Format String	40
Ataques Buffer Overflow	412
Ataques CRLF	327

Fonte: (CAPPO, 2017, p. 77).

As bases para este experimento foram configuradas da seguinte forma: o conjunto de dados normais foi dividido em dois conjuntos, de treino e de teste com 70% e 30%, respectivamente. Ao conjunto de treino foram adicionadas 50 amostras do tipo de ataque. A mesma quantidade foi adicionada ao conjunto de teste. Os ataques inseridos nos conjuntos de treino e de testes são distintos e escolhidos aleatoriamente na base CSIC 2012.

Desta forma, para este experimento, a configuração dos dados ficou definida como segue, sendo os 50 ataques sempre de um mesmo tipo, mas sem repetí-los.

- Conjunto de Treino: Normal=5854 Attacks=50
- Conjunto de Teste: Normal=2509 Attacks=50.

A disparidade entre a quantidade de amostras normais e amostras de ataques no conjunto de teste faz com que a métrica de acurácia não seja apropriada para a comparação entre as redes. Com uma quantidade tão baixa, mesmo que modelo não detectasse nenhum ataque ainda teríamos mais 98% de acurácia. A pontuação F1 é uma métrica mais adequada para este comparativo por se tratar de uma medida que calcula a média ponderada entre a Precisão (P) e da Taxa de Detecção (DR). A medida F1 alcança sua melhor pontuação no valor 1 quando a P e DR são perfeitas. O seu cálculo é definido pela Equação 6.2.

$$F1 = \frac{2 * DR * P}{DR + P} \quad (6.2)$$

A tabela 9 apresenta os resultados do experimento para as métricas TP, FP e F1. Cabe destacar que as redes operam sobre nove atributos (vide seção 4.2.2) e que a mudança destes atributos podem gerar mudanças nestes resultados.

Tabela 9 – Resultados de detecção por tipo de ataque aplicado às redes neurais recorrentes LSTM, Bi-LSTM e GRU.

Attack (qtde)	BLOOM + LSTM			BLOOM + BI-LSTM			BLOOM + GRU		
	TP	FP	F1	TP	FP	F1	TP	FP	F1
XSS	46	3	0.929	39	5	0.830	33	1	0.786
SQLi	45	5	0.900	40	6	0.833	36	2	0.818
XPath	47	0	0.969	48	0	0.980	48	0	0.980
SSI	36	0	0.837	42	0	0.875	29	0	0.734
BufferOverflow	48	1	0.970	48	1	0.970	49	0	0.990
CRLF	45	1	0.938	39	1	0.867	47	1	0.940

De acordo com os resultados tabelados, a rede LSTM apresentou bons índices em todas as categorias de ataques. Apenas para o SSI acusou índice inferior aos 90% (0.837), resultado que também se repetiu nas outras redes recorrentes. A rede LSTM teve a maior pontuação F1 nas categorias de ataques XSS, SQLi e CRLF. A rede GRU obteve baixo índice de falso positivo em todos os ataques verificados. Esta rede também obteve a maior pontuação F1 entre todas as redes e ataques com 0.99 para a categoria *Buffer Overflow* e também foi superior nos ataques Xpath e CRFLi. Já a

rede Bidirecional oscilou entre índices bons (XPath=0.98) e razoáveis (XSS=0.83). Foi melhor para os ataques SSI, BufferOverflow e CRLF.

O experimento demonstra que o modelo BLOOM-RNN, quando aplicado com redes LSTM possibilita obter um desempenho adequado para praticamente todas as categorias de ataques consideradas. Da mesma forma, demonstra que, se especializado por tipo de ataque, pode alcançar desempenho próximo a 100% (F1 igual ou superior a 0.98).

6.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo os experimentos foram organizados para avaliar o desempenho de três redes neurais recorrentes quando aplicadas ao modelo proposto neste trabalho. Foram realizados dois experimentos precedidos por uma etapa de ajustes dos hiperparâmetros das redes neurais recorrentes. Os dados resultantes desta etapa são importantes, pois, não há na literatura pesquisada (Seção 2.2) a exploração destes hiperparâmetros para redes neurais recorrentes aplicadas ao domínio de aplicações da Web.

No primeiro experimento o principal objetivo foi comparar a eficácia preditiva e eficiência das três redes neurais recorrentes propostas. As redes foram avaliadas e foram apresentadas as medidas preditivas de Acurácia, Precisão e Taxa de Detecção. Ao considerar apenas o aspecto da precisão preditiva a rede BI-LSTM tem vantagem sobre suas concorrentes. Ela apresentou a melhor acurácia e precisão, e seu índice para a taxa de detecção é muito próximo do maior valor, obtido pela GRU. Por outro lado, a GRU é mais eficiente em tempo médio de detecção. Sendo assim, a medida ARR, que combina informações sobre a acurácia (ACC) e o tempo de (AVG) foi aplicada para exprimir um resultado geral do modelo. A rede neural BI-LSTM obteve a maior pontuação ARR, seguida por GRU e LSTM. Apesar ter o maior tempo médio de detecção que suas concorrentes, a BI-LSTM consegue se manter competitiva graças aos bons índices preditivos. Com isso é possível estabelecer que a rede neural BI-LSTM como a melhor opção para o modelo proposto neste trabalho.

O segundo experimento focou apenas no desempenho preditivo de cada rede neural diante de tipos específicos de ataques presentes em uma versão atualizada do conjunto de dados CSIC. Neste cenário, a rede neural LSTM tem desempenho mais equilibrado alcançando pontuações (F1) acima de 0.90 para quase todos os ataques. Para as detecções da categoria SSI todas as redes recorrentes tiveram baixo desempenho o que significa que os atributos escolhidos não atenderam bem a esta categoria. Em contrapartida, todas as redes tiveram boa pontuação para os ataques XPath e BufferOverflow mostrando que os recursos escolhidos atendem melhor a estes ataques. Por fim, não há entre os trabalhos relacionados do Capítulo 2 pesquisa que analise o desempenho de seus modelos de detecção com categorias de ataques separadamente

o que representa um mérito adicional para os resultados deste experimento.

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi proposto um modelo de detecção de ataques contra aplicações da Web. Nele, o Filtro de Bloom é combinado com Rede Neural Recorrente para realizar uma classificação binária, ou seja, apontar se uma requisição é um ataque ou uma solicitação normal. O objetivo foi apresentar uma solução baseada em Redes Neurais Recorrentes que atenda a necessidade de se projetar proteções dinâmicas que aprendam constantemente com a identificação de novos ataques. A seguir são apresentadas as principais conclusões da pesquisa realizada, bem como alguns trabalhos que podem ser realizados para investigar em mais profundidade o tema.

7.1 CONCLUSÕES

O modelo proposto neste trabalho apropria-se das características do Filtro de Bloom para reduzir o volume de dados enviados a um detector de ataques web baseado em redes neurais recorrentes. A combinação do Filtro ao detector é a principal contribuição deste trabalho. Os resultados mostraram que o Filtro de Bloom reduz significativamente o volume de dados e conseqüentemente o tempo de detecção quando usado como ferramenta de filtragem. A filtragem ultrapassou os 50% e a redução do tempo médio de detecção foi reduzido para menos da metade nas sete técnicas testadas. Além disso, sua presença no modelo BLOOM-RNN não reduziu a acurácia da detecção de ataques web com redes recorrentes, tão pouco com outros modelos baseados em aprendizado de máquina, tendo inclusive acrescido a acurácia, precisão e taxa de detecção, mesmo que em pequenos níveis.

Cabe o destacar o efeito do filtro diante dos demais classificadores testados, especialmente aqueles que tiveram baixo desempenho sem o filtro. Os classificadores que apresentaram acurácia inferior aos 90% foram os que tiveram maior ganho com a presença do filtro. Os classificadores *Decision Tree* (DT), *Multilayer Perceptron* (MLP) e *Random Forest* (RF) tiveram ganhos expressivos. Este ganho se deu principalmente pela redução de requisições que são enviadas a eles, fruto da filtragem do Bloom. Ao identificar a maior parte das requisições normais no primeiro estágio do modelo reduziu-se a incidência de falsos positivos, aumentando a acurácia.

A aplicação do Filtro de Bloom de tamanho expandido também representa uma importante contribuição deste trabalho. Os experimentos adicionais onde se explorou a configuração do filtro permitiram descobrir benefícios na expansão do filtro. Ao configurar o Bloom com duas vezes o tamanho padrão conseguiu-se eliminar a incidência de falsos negativos causados pelas colisões do filtro de tamanho padrão.

No que tange ao tempo de detecção a influência no modelo é exclusiva do Filtro de Bloom. Sua presença no modelo o torna mais eficiente quanto mais requisições forem filtradas. Sua combinação com técnicas de aprendizado profundo, como é o

caso das redes recorrentes, se mostrou adequado, pois, estas são projetadas para manipular grande quantidade de dados.

Os cenários comparativos entre as redes neurais recorrentes testadas permitiram avaliar o comportamento de cada uma delas sob três óticas: qual rede tem o melhor desempenho preditivo, qual é a mais ágil e, qual consegue equilibrar melhor essas métricas. Pela arquitetura similar das três redes propostas, era de se esperar acurácia semelhante, mas com leve vantagem para a rede BI-LSTM. Da mesma forma, se espera maior agilidade da rede GRU por sua arquitetura mais simples. Previsões que se confirmaram nos resultados dos experimentos comparativos, apesar de, o tempo médio de detecção da rede bidirecional se equiparar à LSTM, largamente preferida nos trabalhos recentes. Ao medir o desempenho entre as redes de forma a considerar tempo e predição verificou-se a melhor adequação da BI-LSTM para o modelo. Seus índices preditivos superam a leve desvantagem em relação ao tempo.

Um segundo cenário comparativo avaliou o desempenho das redes recorrentes diante de variados ataques presentes no conjunto de ataques CSIC 2012. Com foco apenas no desempenho preditivo de cada rede neural, seis categorias de ataques foram testadas. Ao especializar o modelo, a rede neural LSTM teve destaque por conseguir bons índices de detecção para quase todos os ataques. A especialização do modelo por tipo de ataque permite especular a adequação dos atributos escolhidos para representar requisições no modelo. Fica evidente que os nove atributos não representam bem os ataques do tipo SSI e, por outro lado, atendem a ataques do tipo *Buffer Overflow* e XPath. A expansão desses atributos poderia melhorar a precisão nos contextos específico e geral do modelo BLOOM-RNN.

Sendo assim, este trabalho explanou o potencial da combinação entre o Filtro de Bloom e as Redes Neurais Recorrentes. O uso do Filtro de Bloom mostrou ser uma excelente ferramenta para reduzir os dados enviados ao detector baseado em rede neural, reduzindo tempo médio de detecção e mantendo os índices de predição dos classificadores que o acompanharam. O bom desempenho das redes neurais recorrentes para detecção com grandes quantidades de dados também foi confirmado, especialmente o da rede neural recorrente Bidirecional quando aplicado à detecção de ataques Web.

7.2 TRABALHOS FUTUROS

Para a continuidade das pesquisas sugere-se especializar o modelo BLOOM-RNN para alcançar melhores índices preditivos para variadas categorias de ataques de forma a permitir um modelo genérico, mas eficiente e eficaz. A ampliação dos nove atributos pode trazer melhorias neste sentido, mas também pode afetar o modelo do ponto de vista do tempo em caso de uma ampliação demasiada. Outro ponto que pode ser explorado para melhoria do modelo BLOOM-RNN é a dinâmica do filtro. Pode-se

alternar os dados que o preenchem de normais para maliciosos e inverter a atuação do filtro que funcionaria como uma lista de assinatura de ataques. Pode-se ainda, manter o filtro como foi apresentado neste trabalho e adicionar um segundo filtro com assinaturas de ataques após a rede neural para aferir as requisições após a classificação.

É sugerido também realizar experimentos que façam a aplicação do modelo BLOOM-RNN em aplicação Web em ambiente real de produção. Estes testes trariam dados mais precisos sobre tempo de detecção ao acrescentar o tempo de resposta de uma aplicação real ao tempo de detecção do modelo.

A combinação do Filtro de Bloom com redes neurais recorrentes também pode ser explorada em outros domínios como, por exemplo, no reconhecimento de mensagens de 'spam' em aplicações de correio eletrônico. Este desafio em particular traria a necessidade de selecionar atributos específicos para cada aplicação, tal como, representar uma mensagem de *Spam*.

Referências

- ABRAHAM, A. et al. Machine learning for neuroimaging with scikit-learn. **Frontiers in neuroinformatics**, Frontiers, v. 8, 2014. Citado 2 vezes nas páginas 43 e 52.
- AHMED, M.; MAHMOOD, A. N.; HU, J. A survey of network anomaly detection techniques. **Journal of Network and Computer Applications**, v. 60, p. 19 – 31, 2016. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804515002891>. Citado na página 12.
- AL-JARRAH, O. Y. et al. Efficient Machine Learning for Big Data: A Review. **Big Data Research**, v. 2, n. 3, p. 87 – 93, 2015. ISSN 2214-5796. Big Data, Analytics, and High-Performance Computing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2214579615000271>. Citado na página 41.
- ALTHUBITI, S. et al. Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection. In: **SoutheastCon 2018**. [S.l.: s.n.], 2018. p. 1 – 5. ISSN 1558-058X. Citado 3 vezes nas páginas 17, 35 e 52.
- ALTHUBITI, S.; YUAN, X.; ESTERLINE, A. Analyzing HTTP requests for web intrusion detection. 2017. Citado 6 vezes nas páginas 12, 15, 16, 35, 41 e 44.
- ALTHUBITI, S. A.; JONES, E. M.; ROY, K. LSTM for Anomaly-Based Network Intrusion Detection. In: **2018 28th International Telecommunication Networks and Applications Conference (ITNAC)**. [S.l.: s.n.], 2018. p. 1 – 3. Citado na página 25.
- BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. **Communications of the ACM**, ACM, v. 13, n. 7, p. 422 – 426, 1970. Citado 3 vezes nas páginas 13, 30 e 37.
- BOCHEM, A.; ZHANG, H.; HOGREFE, D. Streamlined anomaly detection in web requests using recurrent neural networks. In: IEEE, 2017. **2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.], 2017. p. 1016 – 1017. Citado 4 vezes nas páginas 12, 16, 41 e 52.
- BRAZDIL, P. B.; SOARES, C.; COSTA, J. P. D. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. **Machine Learning**, Springer, v. 50, n. 3, p. 251 – 277, 2003. Citado na página 57.
- BRODER, A.; MITZENMACHER, M. Network applications of bloom filters: A survey. **Internet mathematics**, Taylor & Francis, v. 1, n. 4, p. 485 – 509, 2004. Citado na página 30.
- CAPPO, C. **Detección de ataques en Aplicaciones Web aplicando la Transformada Wavelet**. 2017. 128 p. Tese (Doctorado en Ciencias de la Computación) — Universidad Nacional de Asunción, San Lorenzo. Citado 4 vezes nas páginas 22, 23, 24 e 34.
- CHOLLET, F. **Keras: The Python Deep Learning library**. Disponível em: <https://keras.io/>. 2015. Citado na página 43.
- CLAESEN, M.; MOOR, B. D. Hyperparameter search in machine learning. **arXiv preprint arXiv:1502.02127**, 2015. Citado na página 28.

- CRUZ, E. P. F. da. O uso de filtros de Bloom em protocolo baseado na métrica de ou-exclusivo em redes veiculares urbanas. 2012. Citado na página 30.
- CSIC. **HTTP dataset CSIC 2010**. 2010. Disponível em: <http://www.isi.csic.es/dataset/>. Acesso em: 23/07/2019. Citado 4 vezes nas páginas 15, 18, 42 e 57.
- DATA SCIENCE ACADEMY. **Deep Learning Book**. 2019. Digital. Disponível em: <http://deeplearningbook.com.br/>. Acesso em: 16/10/2019. Citado na página 27.
- EPP, N.; FUNK, R.; CAPPO, C. Anomaly-based Web Application Firewall using HTTP-specific features and One-Class SVM. In: . [S.l.: s.n.], 2017. Citado 2 vezes nas páginas 12 e 15.
- FENG, C.; LI, T.; CHANA, D. Multi-level anomaly detection in industrial control systems via package signatures and lstm networks. In: IEEE, 2017. **2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.], 2017. p. 261 – 272. Citado 2 vezes nas páginas 13 e 37.
- GIMÉNEZ, C. T. et al. **Study of stochastic and machine learning techniques for anomaly-based Web attack detection**. 2015. Tese (Doutorado) — University Carlos III of Madrid, 2015. Citado 4 vezes nas páginas 12, 35, 36 e 41.
- GRAVES, A.; JAITLY, N.; MOHAMED, A. 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. ISSN null. Citado na página 27.
- GROVE, R. F. **Web Based Application Development**. [S.l.]: Jones & Bartlett Publishers, 2009. Citado 2 vezes nas páginas 22 e 37.
- HAO, S.; LONG, J.; YANG, Y. BL-IDS: Detecting Web Attacks Using Bi-LSTM Model Based on Deep Learning. In: SPRINGER, 2019. **International Conference on Security and Privacy in New Computing Environments**. [S.l.], 2019. p. 551 – 563. Citado 4 vezes nas páginas 12, 17, 27 e 52.
- HERRERA-SEMENETS, V. et al. A data reduction strategy and its application on scan and backscatter detection using rule-based classifiers. **Expert Systems with Applications**, v. 95, p. 272 – 279, 2018. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417417307972>. Citado 2 vezes nas páginas 13 e 34.
- HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735 – 1780, 1997. Citado na página 25.
- HORNIK, K.; BUCHTA, C.; ZEILEIS, A. Open-source machine learning: R meets Weka. **Computational Statistics**, Springer, v. 24, n. 2, p. 225 – 232, 2009. Citado na página 16.
- 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. Citado 2 vezes nas páginas 12 e 16.

- JIANG, M. et al. An improved algorithm based on Bloom filter and its application in bar code recognition and processing. **EURASIP Journal on Image and Video Processing**, SpringerOpen, v. 2018, n. 1, 2018. Citado na página 31.
- JUVONEN, A.; SIPOLA, T.; HÄMÄLÄINEN, T. Online anomaly detection using dimensionality reduction techniques for HTTP log analysis. **Computer Networks**, v. 91, p. 46 – 56, 2015. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128615002650>. Citado na página 36.
- KIM, T.; CHO, S. Web traffic anomaly detection using C-LSTM neural networks. **Expert Systems with Applications**, v. 106, p. 66 – 76, 2018. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417418302288>. Citado 3 vezes nas páginas 12, 18 e 52.
- KINGMA, D. P.; BA, J. Adam: A Method for Stochastic Optimization. **CoRR**, abs/1412.6980, 2014. Disponível em: <http://arxiv.org/abs/1412.6980>. Citado 2 vezes nas páginas 17 e 29.
- KRUEGEL, C.; VIGNA, G. Anomaly Detection of Web-based Attacks. **Proceedings of the 10th ACM conference on Computer and communications security**, p. 251 – 261, Outubro 2003. Citado na página 12.
- LE, T.; KIM, J.; KIM, H. An Effective Intrusion Detection Classifier Using Long Short-Term Memory with Gradient Descent Optimization. In: **2017 International Conference on Platform Technology and Service (PlatCon)**. [S.l.: s.n.], 2017. p. 1 – 6. Citado 2 vezes nas páginas 12 e 25.
- LIANG, J.; ZHAO, W.; YE, W. Anomaly-based web attack detection: a deep learning approach. In: ACM, 2017. **Proceedings of the 2017 VI International Conference on Network, Communication and Computing**. [S.l.], 2017a. p. 80 – 85. Citado 2 vezes nas páginas 12 e 52.
- LIANG, J.; ZHAO, W.; YE, W. Anomaly-based web attack detection: a deep learning approach. In: ACM, 2017. **Proceedings of the 2017 VI International Conference on Network, Communication and Computing**. [S.l.], 2017b. p. 80 – 85. Citado na página 12.
- LIANG, J.; ZHAO, W.; YE, W. Anomaly-based web attack detection: a deep learning approach. In: ACM, 2017. **Proceedings of the 2017 VI International Conference on Network, Communication and Computing**. [S.l.], 2017c. p. 80 – 85. Citado na página 18.
- LIANG, Y.; DENG, J.; CUI, B. Bidirectional LSTM: An Innovative Approach for Phishing URL Identification. In: SPRINGER, 2019. **International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**. [S.l.], 2019. p. 326 – 337. Citado 2 vezes nas páginas 27 e 28.
- MAC, H. et al. Detecting Attacks on Web Applications using Autoencoder. In: ACM, 2018. **Proceedings of the Ninth International Symposium on Information and Communication Technology**. [S.l.], 2018. p. 416 – 421. Citado 2 vezes nas páginas 15 e 35.

- NGUYEN, H. T. et al. Application of the generic feature selection measure in detection of web attacks. In: **Computational Intelligence in Security for Information Systems**. [S.l.]: Springer, 2011. p. 25 – 32. Citado na página 35.
- ONEY, M. U.; PEKER, S. The Use of Artificial Neural Networks in Network Intrusion Detection: A Systematic Review. In: . [S.l.: s.n.], 2018. p. 1 – 6. Citado 2 vezes nas páginas 12 e 41.
- OWASP. **Server-Side Include (SSI) Injection**. Disponível em: [https://owasp.org/www-community/attacks/Server-Side_Includes_\(SSI\)_Injection](https://owasp.org/www-community/attacks/Server-Side_Includes_(SSI)_Injection). Acesso em: 16/01/2020. Citado na página 24.
- OWASP, T. Top 10-2017 The Ten Most Critical Web Application Security Risks. **URL: [owasp.org/images/7/72/OWASP_Top_10-2017_%28en](https://owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf)**, v. 29, 2017. Citado na página 22.
- PARTHASARATHY, S.; KUNDUR, D. Bloom filter based intrusion detection for smart grid SCADA. In: IEEE, 2012. **Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on**. [S.l.], 2012. p. 1 – 6. Citado na página 38.
- PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of machine learning research**, v. 12, n. Oct, p. 2825 – 2830, 2011. Citado 2 vezes nas páginas 52 e 53.
- REIMERS, N.; GUREVYCH, I. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. **arXiv preprint arXiv:1707.06799**, 2017. Citado 3 vezes nas páginas 28, 29 e 53.
- RUWASE, O.; LAM, M. S. A Practical Dynamic Buffer Overflow Detector. In: **NDSS**. [S.l.: s.n.], 2004. v. 2004, p. 159 – 169. Citado na página 23.
- SAHIN, M.; SOGUKPINAR, I. An efficient firewall for web applications (EFWA). In: IEEE, 2017. **2017 International Conference on Computer Science and Engineering (UBMK)**. [S.l.], 2017. p. 1150 – 1155. Citado 2 vezes nas páginas 15 e 41.
- SKARUZ, J.; SEREDYNSKI, F. Recurrent neural networks towards detection of SQL attacks. In: **2007 IEEE International Parallel and Distributed Processing Symposium**. [S.l.: s.n.], 2007. p. 1 – 8. ISSN 1530-2075. Citado na página 23.
- SMITHA, R.; HAREESHA, K.; KUNDAPUR, P. P. A Machine Learning Approach for Web Intrusion Detection: MAMLS Perspective. In: **Soft Computing and Signal Processing**. [S.l.]: Springer, 2019a. p. 119 – 133. Citado na página 12.
- SMITHA, R.; HAREESHA, K.; KUNDAPUR, P. P. A Machine Learning Approach for Web Intrusion Detection: MAMLS Perspective. In: **Soft Computing and Signal Processing**. [S.l.]: Springer, 2019b. p. 119 – 133. Citado na página 16.
- SU, Z.; WASSERMANN, G. The Essence of Command Injection Attacks in Web Applications. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 41, n. 1, p. 372 – 382, jan 2006. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/1111320.1111070>. Citado na página 23.

SYMANTEC. ISTR - Internet Security Threat Report. v. 24, Fevereiro/2019. Disponível em: <https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf>. Acesso em: 16/10/2019. Citado na página 22.

TECHNOLOGIES, P. Web Application Vulnerabilities: Statistics for 2018. March 2019. Disponível em: <https://www.ptsecurity.com/ww-en/analytics/web-application-vulnerabilities-statistics-2019/>. Acesso em: 16/10/2019. Citado na página 22.

TORRANO-GIMENEZ, C.; PEREZ-VILLEGAS, A.; ALVAREZ, G. TORPEDA: Una Especificacion Abierta de Conjuntos de Datos para la Evaluacion de Cortafuegos de Aplicaciones Web. 2012. Citado 4 vezes nas páginas 19, 52, 53 e 57.

WANG, J.; ZHOU, Z.; CHEN, J. Evaluating CNN and LSTM for Web Attack Detection. In: ACM, 2018. **Proceedings of the 2018 10th International Conference on Machine Learning and Computing**. [S.l.], 2018. p. 283 – 287. Citado 2 vezes nas páginas 18 e 52.

WATSON, D. Web application attacks. **Network Security**, Elsevier, v. 2007, n. 10, p. 10 – 14, 2007. Citado na página 24.

YU, Y. et al. Attention-Based Bi-LSTM Model for Anomalous HTTP Traffic Detection. In: IEEE, 2018. **2018 15th International Conference on Service Systems and Service Management (ICSSSM)**. [S.l.], 2018. p. 1 – 6. Citado 3 vezes nas páginas 17, 27 e 52.

ZHANG, M.; LU, S.; XU, B. An Anomaly Detection Method Based on Multi-models to Detect Web Attacks. In: **2017 10th International Symposium on Computational Intelligence and Design (ISCID)**. [S.l.: s.n.], 2017. v. 2, p. 404 – 409. Citado na página 15.

ZHAO, J. et al. Classifying Malicious URLs Using Gated Recurrent Neural Networks. In: SPRINGER, 2018. **International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**. [S.l.], 2018. p. 385 – 394. Citado 2 vezes nas páginas 17 e 52.

ZHOU, L. et al. Machine learning on big data: Opportunities and challenges. **Neurocomputing**, v. 237, p. 350 – 361, 2017. ISSN 0925-2312. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925231217300577>. Citado 3 vezes nas páginas 32, 39 e 41.

ZHOU, W. et al. Detection and defense of application-layer DDoS attacks in backbone web traffic. **Future Generation Comp. Syst.**, v. 38, p. 36 – 46, 2014. Disponível em: <http://dx.doi.org/10.1016/j.future.2013.08.002>. Citado 2 vezes nas páginas 13 e 37.

ZULKIFLI, H. **Understanding Learning Rates and How It Improves Performance in Deep Learning**. 2018. Site da internet. Disponível em: <https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10>. Acesso em: 20/11/2019. Citado na página 29.