Filtro de Bloom como Ferramenta de Apoio a Detectores de Ataques Web baseados em Aprendizagem de Máquina

Richard Caio Silva Rego [1], **Raul Ceretta Nunes** [2]

[1] PPGCC/CT – Universidade Federal de Santa Maria
Av. Roraima, 1000, Camobi, Santa Maria, RS, CEP 97.105-900

[2] PPGCC/CT/Depto de Computação Aplicada – Universidade Federal de Santa Maria
Av. Roraima, 1000, Camobi, Santa Maria, RS, CEP 97.105-900

caiorego@protonmail.com, ceretta@inf.ufsm.br

Abstract. *Attacks against Web applications imply social and financial losses. Current detection systems that use machine learning techniques are not scalable enough to handle large volumes of data. The Bloom Filter is a simple and efficient random data structure that allows you to test whether a particular element belongs to a set of probabilistic shapes. In this paper the Bloom Filter was combined with seven machine learning techniques commonly used in anomaly-based detectors of web attacks. The results show the use of the filter as the first stage of the anomaly-based detection mechanism can reduce both average and total detection time in all tested machine learning classifiers for web attack detectors. The results also show the filter helps to increase the detection accuracy and precision if used a proposed key Bloom Filter setting optimization to mitigate unwanted false negatives.*
Palavras-chave: *Bloom Filter, Web Attacks, Intrusion Detection Systems, Machine Learning.*

Resumo. *Ataques contra aplicações da Web implicam em prejuízos sociais e financeiros. Os sistemas de detecção atuais que utilizam técnicas de aprendizagem de máquina não são escaláveis o suficiente para lidar com grandes volumes de dados. O Filtro de Bloom é uma estrutura de dados aleatória simples e eficiente que permite testar se um determinado elemento pertence a um conjunto de forma probabilística. Neste artigo aplicou-se o Filtro de Bloom combinado com sete técnicas de aprendizagem de máquina comumente utilizadas em detectores de anomalias para ataques web. Os resultados demonstram que o uso do filtro como primeiro estágio do mecanismo de detecção de anomalias reduz tanto o tempo médio quanto o tempo total de detecção em todas as técnicas. Os resultados também demonstram que o filtro pode auxiliar inclusive a incrementar a acurácia e a precisão, se adotada a otimização proposta na configuração do Filtro de Bloom para redução de falsos negativos.*
Palavras-chave: *Filtro de Bloom, Ataques web, Sistemas de Detecção de Intrusão, Aprendizado de Máquina.*

1. Introdução

Ataques contra aplicações da Web podem causar sérios danos ao funcionamento dos serviços, acarretando prejuízos sociais e financeiros [Ahmed *et al.*, 2016]. Um ataque pode

implicar em acesso indevido ao banco de dados, modificação de páginas e indisponibilidade de serviços, entre outros. Novas vulnerabilidades em aplicações da Web surgem todos os dias. Isso exige que as aplicações tenham constantes avanços nos mecanismos de defesa.

Um sistema de detecção de intrusões (SDI) é uma ferramenta de vigilância, no nível de rede ou de *host*, que monitora os eventos e identifica os comportamentos que possam comprometer a integridade, confidencialidade ou disponibilidade de um determinado recurso [Kevric *et al.*, 2016]. Os Firewalls de Aplicações da Web (*Web Application Firewalls* - WAFs) são casos especiais de SDIs especializados em analisar o tráfego HTTP com o objetivo de proteger aplicativos da Web [Giménez *et al.*, 2015].

Os sistemas de prevenção de ataques geralmente usam métodos baseados em assinatura ou anomalias para sistemas de detecção de intrusão. A utilização de técnicas de aprendizado de máquina para detecção de anomalias vem sendo bastante explorada. Sahin e Sogukpınar [2017] aplicaram algoritmos de Árvore de decisão (C4.5) e K-NN para previnir ataques Web. Ito e Iyatomi [2018] utilizaram redes neurais convolucionais para construir um sistema de identificação de requisições HTTP maliciosas. Althubiti *et al.* [2017] classificaram tráfego HTTP como normal ou anômalo aplicando técnicas de aprendizado de máquina como *Random Forest*, *Logistic Regression*, *Decision Tree*, SGD, *Adaboost* e *Naive Bayes*.

A vantagem dos sistemas de detecção baseados em aprendizado de máquina é que detectam ou categorizam recursos persistentes sem nenhum *feedback* do ambiente [Nene e Singh, 2013]. No entanto, os atuais sistemas baseados em aprendizado de máquina não são inerentemente eficientes ou escaláveis o suficiente para lidar com grandes volumes de dados [Al-Jarrah *et al.*, 2015]. Além disso, no aprendizado de máquina inteligente atual os sistemas são orientados pelo desempenho onde o foco está na precisão preditiva de classificação e um pequeno incremento na precisão custa muito tempo computacional [Oney e Peker, 2018]. Para que estes problemas não se sobreponham às qualidades das máquinas de aprendizagem faz-se necessário o uso de estratégias de pré-processamento para reduzir a carga de dados que é enviada ao detector sem afetar a precisão. 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 e no incremento da precisão. Em áreas similares o Filtro de Bloom aparece como opção para fazer esta filtragem [Feng *et al.*, 2017; Zhou *et al.*, 2014].

O Filtro de Bloom é uma estrutura de dados aleatória simples e eficiente em memória e tempo. Ele pode representar um conjunto de elementos e suportar consultas de associação e tem sido utilizado no campo da segurança em bancos de dados, corretores ortográficos, roteamento de recursos e *web cache*, entre outros [Geravand e Ahmadi, 2013]. O Filtro de Bloom tem recebido atenção também na área de redes [Broder e Mitzenmacher, 2004]. Parthasarathy e Kundur [2012] destacam algumas vantagens do filtro em uma estrutura de detecção de anomalias. Suas características o credenciam para anteceder SDIs baseados em aprendizado de máquinas reduzindo o fluxo de requisições no detector.

Neste artigo aplica-se o Filtro de Bloom combinado com sete técnicas de aprendizado de máquina comumente utilizadas em detectores de anomalias (Rede Neural Artificial, *Support Vector Machine*, *Multilayer Perceptron*, *K-Nearest Neighbor*, Rede

Neural Recorrente). A contribuição é a exploração da aplicabilidade do Filtro de Bloom como primeiro estágio de detectores para fins de redução de fluxo das requisições sem afetar a precisão preditiva. Como resultado, demonstrou-se que a presença do filtro reduz o tempo médio e o tempo total de detecção para todas as técnicas investigadas, além de incrementar a acurácia e a precisão da detecção das anomalias geradas por ataques web. O trabalho contribui também demonstrando otimizações nas configurações do Filtro de Bloom que permitem reduzir os casos de falsos negativos.

O artigo está organizado da seguinte forma: Na seção 2 são apresentados os trabalhos atuais que contribuíram para o conhecimento relacionado e o desenvolvimento dos experimentos propostos. A seção 3 apresenta as especificidades e o funcionamento do Filtro de Bloom. A seção 4 apresenta a estrutura de detecção combinando o filtro com os classificadores escolhidos para os testes. A seção 5 apresenta o planejamento, o conjunto de dados e os recursos selecionados para treinamento dos classificadores. Também são descritos os detalhes da implementação dos classificadores. Na seção 6 são apresentadas as considerações finais com a conclusão do trabalho.

2. Trabalhos relacionados

Nesta seção são apresentados trabalhos atuais que envolveram a redução de dados em detectores de anomalias e que contribuem para o desenvolvimento do experimento proposto neste trabalho. Em [Zhou *et al.*, 2014] os autores propõem um novo método para detecção de ataques de negação de serviços da camada de aplicação (AL-DDoS) para filtrar o tráfego malicioso nos *backbones* e impedir danos aos servidores da web. A arquitetura é composta por um módulo de detecção de ataques e um módulo de filtro. Quando o tráfego anormal é comprovadamente um ataque AL-DDoS o módulo de detecção envia os endereços IPs da fonte maliciosa para módulo de filtragem para cessar o ataque. Para armazenar uma lista tão grande de endereços IPs comprometidos é utilizado um filtro de Bloom. Contudo, a solução foca apenas em filtros para IPs que podem estar realizando ataques AL-DDoS e não abrange outras ameaças às aplicações Web.

No trabalho apresentado por Feng *et al.* [2017] os autores descrevem um método de detecção de anomalias multinível específico para sistemas de controle industrial. Sua estrutura é composta de um filtro de Bloom que armazena o banco de dados de assinatura para comportamentos normais de pacotes de rede seguido de uma rede neural LSTM. O uso do filtro de Bloom, neste caso, se deu por haver memória e recursos de computação limitados nos monitores de tráfego de rede presente nos sistemas de controle industrial.

Em [Herrera-Semenets *et al.*, 2018] os autores propõem uma estratégia para a redução de dados que são utilizados no treinamento de classificadores supervisionados. A estratégia combina redução de recursos com redução de instâncias e assim obtém um conjunto de treinamento reduzido melhorando a eficiência do classificador sem afetar muito a precisão. O processo de redução de instâncias, no entanto, é executado apenas sobre a coleção de treinamento o que não altera o processamento e tempo do detector.

Da mesma forma, em [Li *et al.*, 2009] os pesquisadores utilizaram a seleção de instâncias baseada no algoritmo Fuzzy C-Means (FCM), uma técnica de agrupamento separada de k-means que emprega particionamento rígido, para remover os dados normais irrelevantes do conjunto de treinamento. Os experimentos mostraram que a seleção de instâncias reduziu o conjunto de dados de treinamento em quase 90% e o tempo de detecção

em 77,1%. Apesar do bom desempenho o trabalho voltou-se para a detecção de anomalias de rede.

Na detecção de anomalias em ambiente da Web a estratégia de redução de dados está voltada, em grande parte, para a seleção de recursos ideais para o treinamento dos classificadores [Althubiti *et al.*, 2017; Nguyen *et al.*, 2011; Giménez *et al.*, 2015]. Neste artigo, o principal diferencial é a análise de impactos da aplicação do Filtro de Bloom como primeiro estágio de uma estrutura de detecção de ataques web. Além disto, a exploração de suas variações de configuração para obter um melhor desempenho e reduzir a quantidade de dados enviados ao classificador. A contribuição é um método para redução do fluxo de dados web e a consequente redução do tempo de detecção sem afetar a precisão do classificador.

3. Filtro de Bloom

O filtro de Bloom [Geravand e Ahmadi, 2013] 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.

Para representar um determinado conjunto de elementos $S = \{x1, x2, \ldots, xn\}$ o filtro é descrito como um vetor de tamanho m de bits inicialmente preenchidos com valor 0 (figura 1a). Cada elemento inserido no vetor (array) 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 1b). Para verificar se um item y está no vetor, aplica-se o hash e verifica se todas as posições do elemento y estão configuradas para 1. Se todas as posições apontadas pelos hash de y estiverem definidas como 1, assume-se então que y pode estar no conjunto (*Y1* na figura 1c). Se não, definitivamente, y não é um membro do conjunto [Broder e Mitzenmacher, 2004] (*Y2* na figura 1c).

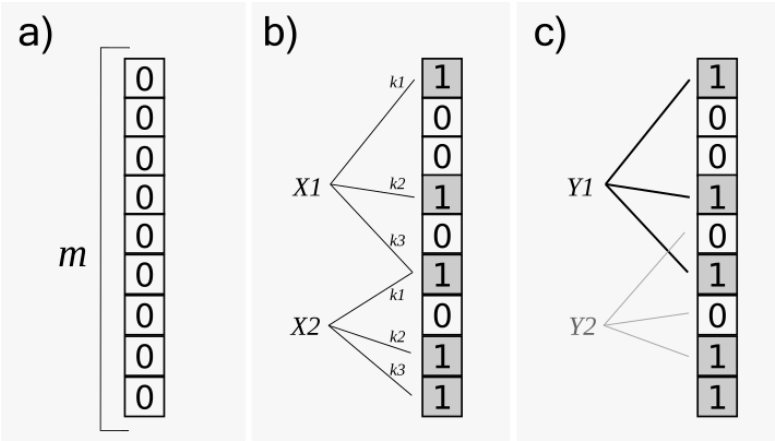

Figura 1. (a) Filtro vazio (b) Preenchimento do filtro (c) Consulta de elementos no filtro.

Há uma compensação entre os requisitos de memória do filtro de Bloom e a taxa

de falsos positivos. Esta taxa pode ser ajustada através da escolha dos parâmetros m e k. Se a quantidade de elementos 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 1). Conhecendo o tamanho do vetor m e a quantidade de itens n é possível calcular o número ideal de funções hash utilizadas para as operações de inserção e consulta de elementos do filtro (equação 2).

$$m = \frac{n \times ln(P)}{ln^2(2)} \tag{1}$$

$$K = \frac{m}{n \times ln(2)} \tag{2}$$

Observa-se que o filtro de Bloom é eficiente em espaço comparado a outras estruturas de dados, como matrizes, listas vinculadas e tabelas de hash, essencial para aplicações da Web.

4. Detector de anomalia combinado com filtro de Bloom

Os algoritmos de aprendizado de máquina tradicionais enfrentam desafios críticos de escalabilidade para grandes volumes de dados [Zhou *et al.*, 2017]. Além disso, as técnicas de aprendizado profundo demonstraram grande desempenho em tarefas de reconhecimento e classificação e têm sido preferidas para detecção de anomalias em muitos trabalhos especialmente a partir de 2017 [Oney e Peker, 2018]. No entanto, o aprendizado profundo possui alto custo computacional e dificuldades de escalabilidade [Al-Jarrah *et al.*, 2015].

Neste trabalho, para avaliar a aplicabilidade do filtro de Bloom como fase inicial de detectores de ataques web, foram utilizados sete classificadores diferentes. Os classificadores escolhidos foram aplicados em abordagens de detecção de ataques Web em trabalhos recentes [Giménez *et al.*, 2015; Sahin e Sogukpınar, 2017; Althubiti *et al.*, 2017; Bochem *et al.*, 2017]. São eles: *Decision Tree* (DT), *K-Nearest Neighbors algorithm* (KNN), *Support Vector Machine* (SVM), *Multilayer Perceptron* (MLP), *Random Forest* (RF), Rede Neural Artificial (RNA) e Rede Neural Recorrente (RNR).

Aplicou-se o Filtro de Bloom combinado com cada um dos detectores de anomalias. 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 menor que no detector. Portanto, a filtragem proporciona uma redução no tempo total do conjunto de teste o que, consequentemente, reduz o tempo médio de detecção por requisição.

5. Experimentos e Resultados

Nesta seção são apresentados os experimentos e seu planejamento, bem como, o conjunto de dados utilizado e a motivação de sua escolha para o trabalho e o resultados obtidos. Além disso, apresentam-se as métricas utilizadas para medir o desempenho dos detectores com e sem a presença do filtro de Bloom.

A função hash utilizada pelo Filtro de Bloom deve ser leve e produzir baixo número de colisões. Em [Parthasarathy e Kundur, 2012] os autores testaram quatro diferentes

funções (FNV, *Sax*, *Jenkins*, *Murmur*) e escolheram a *Murmur Hash* descrevendo-a como leve, rápida e com baixo número de colisões. Por este motivo, optou-se pelo uso da função *Murmur Hash* no filtro de Bloom durante os experimentos deste artigo.

5.1. Planejamento dos Experimentos

Os experimentos foram divididos em três partes para que fosse possível medir os impactos da aplicação do filtro de Bloom ao detector. 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.

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. Esta configuração resulta em maior número de falsos negativos, pois, agrega à taxa as colisões do Filtro de Bloom.

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. Tal exploração mostrou que é possível dobrar o tamanho do filtro sem necessidade de aumentar o número de funções hash.

Para os experimentos foi utilidado um computador com um processador Quad core Intel Core i7-4790, cache de 8192 KB, 7882.9 MB de memória RAM e acelaração GPU GeForce GTX 745, com sistema operacional Ubuntu 18.04 64 bits.

5.2. Conjunto de Dados

Neste trabalho foi utilizado o HTTP DATASET CSIC 2010 [Information Security Institute - Spanish Research National Council, 2010]. Este conjunto de dados foi criado por Giménez *et al.* [2015] e desde então vem sendo utilizado para avaliar novos sistemas de detecção de ataques web [Althubiti *et al.*, 2018; Ito e Iyatomi, 2018; Sahin e Sogukpınar, 2017; Bochem *et al.*, 2017; Liang *et al.*, 2017; Araujo, 2017]. O conjunto de dados do CSIC possui 35992 requisições normais e 24664 anômalas. O conjunto contempla ataques estáticos e dinâmicos, incluindo ataques modernos da Web como *SQL Injection*, *Buffer Overflow*, *Cross Site Scripting*, coleta de informações, *CRLF Injection*, inclusão no servidor e adulteração de parâmetros.

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.

5.3. Seleção de Recursos

A seleção de recursos é o processo de escolha dos atributos mais relevantes para classificar os dados. Atributos de uma requisição Web são informações que a caracterizam como, tamanho da requisição, tamanho do argumento, quantidade de caracteres especiais, dentre outros. Esses atributos auxiliam na detecção de ataques. Por exemplo, o comprimento da requisição pode caracterizar um ataque do tipo estouro de *buffer*.

Tabela 1. Recursos extraídos de uma requisição.

Requisição	http://localhost:8080/tienda1/publico/caracteristicas.jsp?id=2
Tamanho da requisição	62
Tamanho do caminho	36
Nº de letras no caminho	31
Nº de caracteres especiais no caminho	4
Tamanho do argumento	4
Nº de argumentos	1
Nº de letras no argumento	2
Nº de dígitos no argumento	1
Valor máximo do Byte na requisição	117

A escolha dos recursos mais relevantes têm sido alvo de pesquisas [Nguyen *et al.*, 2011; Althubiti *et al.*, 2017; Giménez *et al.*, 2015]. Em [Althubiti *et al.*, 2018] os autores chegaram a quantidade de 9 recursos (ver tabela 1) que consideram mais significativos no conjunto de dados HTTP DATASET CSIC 2010. Utilizou-se estes mesmos recursos para treinar os classificadores utilizados nos experimentos deste trabalho. Cada amostra utilizada nos experimentos continha uma URL pura fornecida no conjunto de dados do CSIC e 9 valores numéricos referentes aos seus recursos extraídos conforme a tabela 1.

5.4. 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].

No Scikit-learn, todos os objetos e algoritmos aceitam dados de entrada na forma de matrizes bidimensionais de amostras de tamanho × recursos [Abraham *et al.*, 2014]. Isso facilitou a implementação uma vez que os dados de treino estão organizados em matriz de dados com e sem recursos numéricos que representam as características das requisições HTTP. 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 quatro camadas sendo 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 *et al.*, 2017], uma camada de *dropout* com 0,2 e uma camada de saída com função de ativação *sigmoid*.

- Rede Neural Artificial - RNA
- Rede Neural Recorrente LSTM - RNR

Todos os classificadores receberam configurações básicas em sua implementação e não houve exploração de configuração dos parâmetros, dado que, alcançar alto níveis de precisão não é o foco principal deste trabalho.

5.4.1. Detecção com Filtro de Bloom (D-BLOOM)

Neste experimento aplicou-se um Filtro de Bloom precedendo os detectores. Há uma compensação entre os requisitos de memória m do filtro de Bloom e a taxa de falsos positivos dos detectores. Esta taxa pode ser ajustada através da escolha dos parâmetros m e k [Parthasarathy e Kundur, 2012], conforme as equações apresentadas na seção 3. Ainda, é importante frizar, que para um detector com filtro (arquitetura Filtro -> Detector) uma colisão no filtro corresponde ao descarte de um amostra que não pertence ao conjunto de amostras de treinamento. Logo, esta "colisão", que ocorre quando o hash de duas entradas diferentes apontam para os mesmos índices do vetor do filtro, eventualmente pode gerar falso negativo no detector com filtro.

Do conjunto de treino foram utilizadas apenas as URLs rotuladas como "normal" para preencher o Filtro de Bloom. Neste processo também foram removidas as URLs repetidas ficando 4399 URLs normais para preencher o filtro. Tendo conhecimento dessa quantidade de elementos foram então calculados o tamanho do filtro (Equação 1) e o número de funções hash (Equação 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 foi, primeiramente, verificada no Filtro de Bloom e somente aquelas ausentes no filtro foram encaminhadas aos detectores que fazem a classificação da URL como normal ou anômala. Amostras presentes no filtro, incluindo as colisões, foram classificadas automaticamente como normais por ter um elemento semelhante no filtro.

5.4.2. Detecção com Filtro de Bloom expandido (D-BLOOM+)

A aplicação do filtro (experimento D-BLOOM) representou uma melhora na acurácia e precisão, bem como, no tempo de detecção. Porém, mesmo configurado para obter uma taxa de colisão de 1% o uso do filtro implicou em aumento de falsos negativos (detalhes na seção 6) decorrente das colisões, o que já era esperado. Deste modo, na terceira parte dos experimentos (D-BLOOM+) foram exploradas possibilidades de configuração do filtro de Bloom para minimizar os falsos negativos gerados pela presença do filtro.

Para este experimento utilizou-se o conjunto de treino, com dados “normais” do *dataset*, para preenchimento do filtro e o conjunto de teste, com dados de ataque, para verificar a presença de dados de ataque no filtro (falsos negativos). Neste teste, espera-se o menor número de colisões possíveis, uma vez que, os dois conjuntos apresentam dados completamente distintos. Com base na configuração obtida a partir das equações 1 e 2 apresentadas na seção anterior, sendo P=0,01, o filtro teria tamanho m=42165 e k=7 para a quantidade de elementos n=4399. Na figura 2 observa-se a combinação de várias configurações para *m e k.*

Observa-se que optar por uma simples redução do valor de P implicaria no aumento do tamanho do filtro e do número de funções *hash*. E aumentar a quantidade de funções hash poderia implicar em aumento no tempo de detecção. Porém, observando a Figura 2 é possível perceber que ao dobrar o tamanho do filtro o número de colisões reduz significativamente, sem necessidade de aumentar as funções hash. Esta análise não foi encontrada em outras literaturas e possibilita a avaliação de um modelo de detecção com um filtro de Bloom expandido, com o dobro de tamanho do aplicado ao experimento anterior. A comparação do D-BLOOM com o D-BLOOM+ possibilita compreender melhor as potencialidades do modelo de detecção com Filtro de Bloom antes dos algoritmos de aprendizagem de máquina.

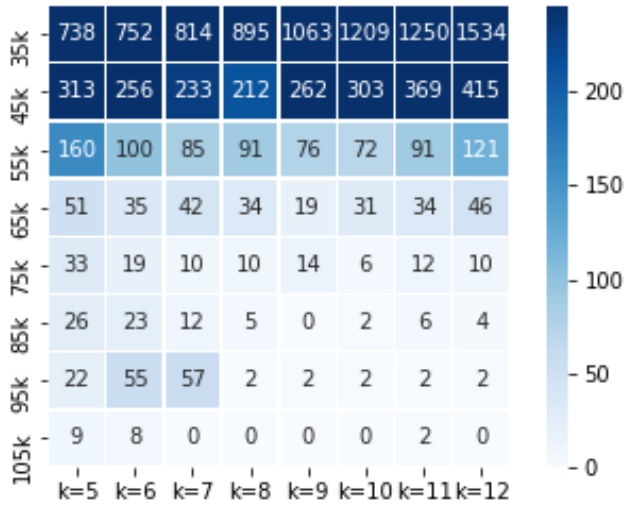

Figura 2. Quantidade de colisões no filtro de acordo com a configuração

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 2 apresenta a configuração do filtro expandido em comparação ao modelo anterior. Salienta-se que a única diferença é 42kbits de tamanho.

Tabela 2. Configuração do Filtro

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

5.5. Métricas

Para avaliar os experimentos propostos calculou-se a acurácia (AC), a precisão (P) e a taxa de detecção (TD) com base em uma matriz de confusão (Tabela 4), sendo: VP (verdadeiro positivo), VN (verdadeiro negativo), FP (falso positivo) e FN (falso negativo). As métricas apresentadas na Tabela 3 foram incluídas para análise preditiva, pois, apesar do desempenho não ser o foco do artigo, a presença do filtro na estrutura de detecção não deve influenciar negativamente no desempenho.

Tabela 3. Métricas

Métrica	Acurácia	Precisão	Taxa de Detecção
Abreviação	AC	P	TD
Fórmula	$\frac{VP+VN}{VP+VN+FP+FN}$	$\frac{VP}{VP+FP}$	$\frac{VP}{VP+FN}$

Tabela 4. Matriz de confusão

	Normal	**Anomalia**
Normal	VN	FP
Anomalia	FN	VP

Além destas métricas de detecção, também foram analisados o *tempo médio e total de detecção.* O *tempo médio* é calculado através da média aritmética dos tempos registrados durante a verificação de cada requisição no filtro e nos classificadores. O *tempo total de detecção* é o período em segundos que a estrutura levou para classificar todos os registros do conjunto de testes, inclusive a consulta ao filtro. Na próxima seção são detalhados os resultados obtidos e a comparação entre os cenários com e sem a presença do Filtro de Bloom como etapa prévia de detecção.

6. Resultados e Discussão

O principal objetivo dos experimentos foi analisar os impactos da aplicação do Filtro de Bloom, como primeiro estágio de uma estrutura de detecção de ataques web. O propósito foi reduzir a quantidade de elementos (URLs) enviados ao detector, diminuindo assim o tempo de detecção da estrutura sem afetar a precisão. Desta forma, analisou-se cada uma das técnicas de aprendizado de máquina individualmente, bem como com o Filtro de Bloom (D-BLOOM) e com o Filtro de Bloom Expandido (D-BLOOM+). Como é

possível observar na Figura 3 a presença do Filtro de Bloom representou aumento, mesmo que singelo, na acurácia e na precisão de todos os classificadores testados. Isto demonstra que o filtro, mesmo reduzindo o número de dados do detector, não degrada a acurácia e precisão podendo até melhorá-la.

A taxa de detecção, no entanto, apresentou queda em todos os classificadores em D-BLOOM. Isto se dá devido às colisões do filtro, que no contexto geral do modelo, representam um falso negativo. Porém, em D-BLOOM+ o filtro elimina estas colisões permitindo manter as taxas de detecção dos classificadores. O resultado demonstra, desta forma, 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 aumento das taxas de precisão e acurácia.

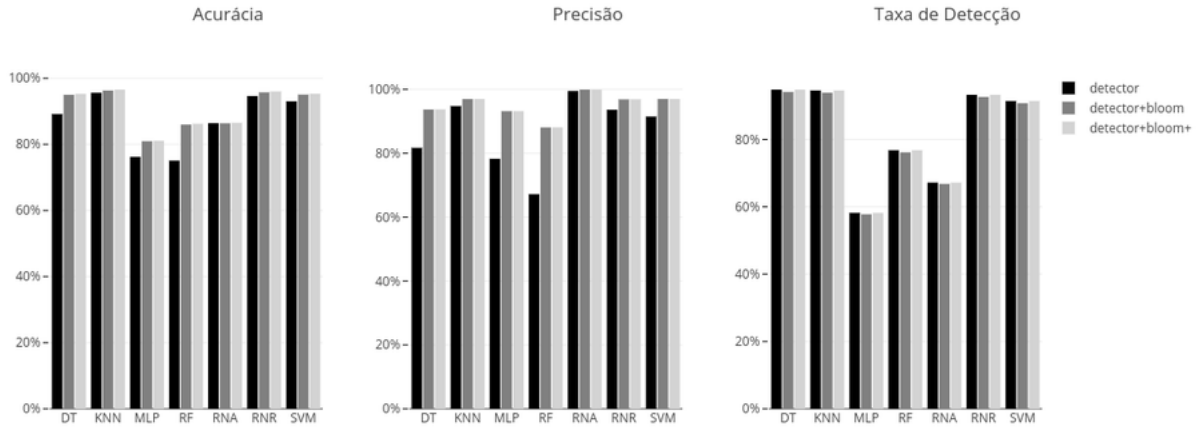

Figura 3. Acurácia, precisão e taxa de detecção dos detectores testados.

Em relação ao tempo de detecção a presença do filtro provocou uma redução no tempo médio que cada requisição leva para passar pela estrutura de detecção. Em geral, essa redução foi de cerca de 50% com Filtro de Bloom padrão e relativamente idêntica com o filtro expandido. A rede neural recorrente testada, que apresenta maior tempo médio (cerca de 1 milisegundo), apresentou uma ótima redução para 0,3 milissegundos com Bloom expandido. Este resultado indica que o filtro tem potencial para apoiar técnicas de detecção computacionalmete pesadas.

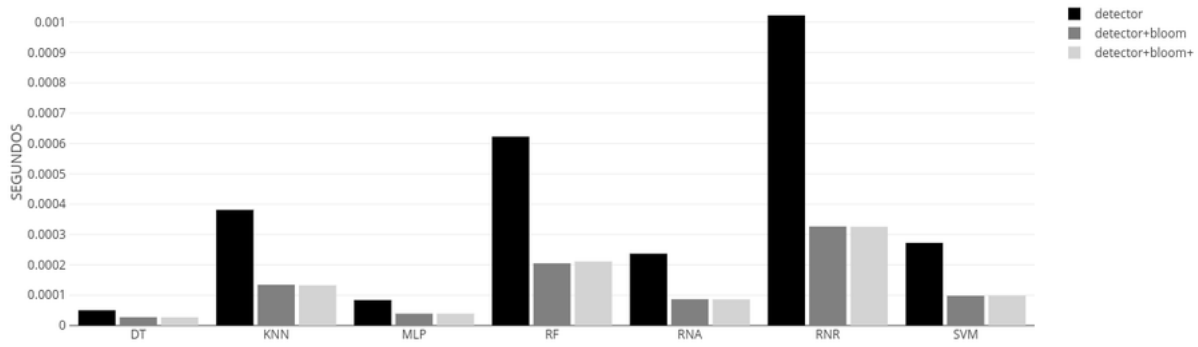

Figura 4. Tempo médio por requisição

O tempo total de detecção também foi reduzido com a presença do filtro para

todos os classificadores testados. Mais uma vez, destaca-se a rede neural recorrente, que levou 18,6 segundos para classificar as 18197 requisições do conjunto de testes. Ela teve esse tempo reduzido para 8,63 segundos com o Filtro de Bloom e 8,62 segundos com o filtro expandido. Importante frisar que, o filtro melhorou os indicadores mesmo dos classificadores que já tinham um tempo reduzido de detecção. O DT, com 0,9 segundos, teve o menor tempo total entre os classificadores e também apresentou melhora com o Filtro de Bloom (0,72) e expandido (0,71). O resultado demonstra o potencial do filtro para otimizar detectores de ataques web baseados em técnicas de aprendizagem de máquina.

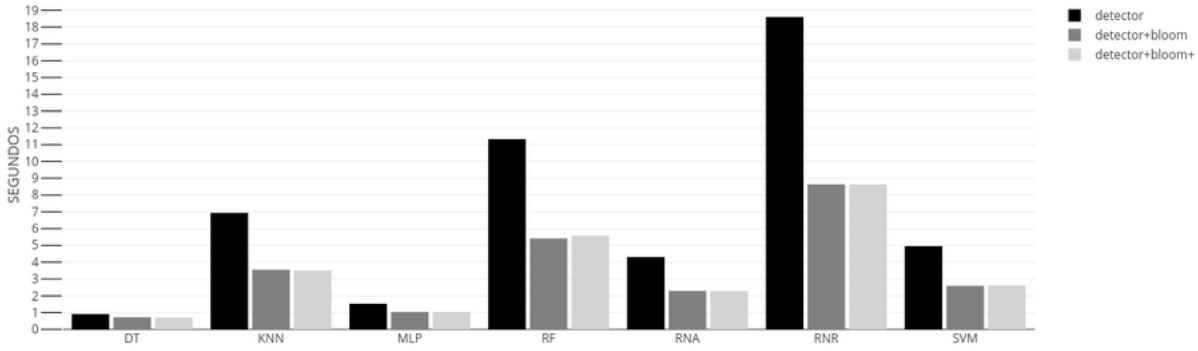

Figura 5. Tempo total de detecção do conjunto de testes

7. Conclusão

Neste artigo 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 demonstrou ser eficaz na redução do tempo médio e do tempo total de detecção em todas as técnicas testadas nos experimentos. Ademais, o Filtro de Bloom apresentou melhoria nas taxas de precisão e acurácia em todos os classificadores sem afetar negativamente a taxa de detecção.

Com foco nos potenciais do Filtro de Bloom neste trabalho não houve exploração dos parâmetros de configuração dos classificadores. Estes ajustes poderão ser explorados em trabalhos futuros onde se acredita que o filtro possa elevar ainda mais os níveis de acurácia e precisão do modelo de detecção com filtro antes do classificador sem comprometer o desempenho do detector de ataques web.

Referências

Abraham, A., Pedregosa, F., Eickenberg, M., Gervais, P., Mueller, A., Kossaifi, J., Gramfort, A., Thirion, B., e Varoquaux, G. (2014). Machine learning for neuroimaging with scikit-learn. *Frontiers in neuroinformatics*, **8**.

Ahmed, M., Mahmood, A. N., e Hu, J. (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, **60**, 19 – 31.

Al-Jarrah, O. Y., Yoo, P. D., Muhaidat, S., Karagiannidis, G. K., e Taha, K. (2015). Efficient Machine Learning for Big Data: A Review. *Big Data Research*, **2**(3), 87 – 93. Big Data, Analytics, and High-Performance Computing.

Althubiti, S., Yuan, X., e Esterline, A. (2017). Analyzing HTTP requests for web intrusion detection.

Althubiti, S., Nick, W., Mason, J., Yuan, X., e Esterline, A. (2018). Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection. In *SoutheastCon 2018*, pages 1 – 5.

Araujo, C. R. C. (2017). Detección de ataques en Aplicaciones Web aplicando la Transformada Wavelet.

Bochem, A., Zhang, H., e Hogrefe, D. (2017). Streamlined anomaly detection in web requests using recurrent neural networks. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1016 – 1017. IEEE.

Broder, A. e Mitzenmacher, M. (2004). Network applications of bloom filters: A survey. *Internet mathematics*, **1**(4), 485 – 509.

Chollet, F. (2015). Keras: The Python Deep Learning library. Disponível em: https://keras.io/.

Feng, C., Li, T., e Chana, D. (2017). Multi-level anomaly detection in industrial control systems via package signatures and lstm networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 261 – 272. IEEE.

Geravand, S. e Ahmadi, M. (2013). Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks*, **57**(18), 4047 – 4064.

Giménez, C. T. *et al.* (2015). *Study of stochastic and machine learning techniques for anomaly-based Web attack detection*. Ph.D. thesis, University Carlos III of Madrid, 2015.

Herrera-Semenets, V., Pérez-García, O. A., Hernández-León, R., van den Berg, J., e Doerr, C. (2018). A data reduction strategy and its application on scan and backscatter detection using rule-based classifiers. *Expert Systems with Applications*, **95**, 272 – 279.

Information Security Institute - Spanish Research National Council (2010). HTTP dataset CSIC 2010. Disponível em: http://www.isi.csic.es/dataset/.

Ito, M. e Iyatomi, H. (2018). Web application firewall using character-level convolutional neural network. In *2018 IEEE 14th International Colloquium on Signal Processing Its Applications (CSPA)*, pages 103 – 106.

Kevric, J., Jukic, S., e Subasi, A. (2016). An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*.

Li, Y., Lu, T., Guo, L., Tian, Z., e Qi, L. (2009). Optimizing Network Anomaly Detection Scheme Using Instance Selection Mechanism. In *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, pages 1 – 7.

Liang, J., Zhao, W., e Ye, W. (2017). Anomaly-based web attack detection: a deep learning approach. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, pages 80 – 85. ACM.

Nene, M. e Singh, J. (2013). A survey on machine learning techniques for intrusion detection systems. *International Journal of Advanced Research in Computer and Communication Engineering*, **11**.

Nguyen, H. T., Torrano-Gimenez, C., Alvarez, G., Petrovic, S., e Franke, K. (2011). Application of the generic feature selection measure in detection of web attacks. In *Computational Intelligence in Security for Information Systems*, pages 25 – 32. Springer.

Oney, M. U. e Peker, S. (2018). The Use of Artificial Neural Networks in Network Intrusion Detection: A Systematic Review. pages 1 – 6.

Parthasarathy, S. e Kundur, D. (2012). Bloom filter based intrusion detection for smart grid SCADA. In *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, pages 1 – 6. IEEE.

Sahin, M. e Sogukpınar, I. (2017). An efficient firewall for web applications (EFWA). In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 1150 – 1155. IEEE.

Zhou, L., Pan, S., Wang, J., e Vasilakos, A. V. (2017). Machine learning on big data: Opportunities and challenges. *Neurocomputing*, **237**, 350 – 361.

Zhou, W., Jia, W., Wen, S., Xiang, Y., e Zhou, W. (2014). Detection and defense of application-layer DDoS attacks in backbone web traffic. *Future Generation Comp. Syst.*, **38**, 36 – 46.