PhishKiller: Uma Ferramenta para Detecção e Mitigação de Ataques de Phishing Através de Técnicas de Deep Learning

Cristian Henrique M. Souza[1], Marcilio O. O. Lemos[1,3], Felipe S. Dantas Silva[1,3], Robinson Luis S. Alves[2]

[1]LaTARC Research Lab
[2]Núcleo de Desenvolvimento de Software
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN)
Av. Sen. Salgado Filho, 1559 – Natal – RN, Brasil

[3]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte (UFRN) – Natal – RN, Brasil

{cristianmsbr, marcilio.cc.lemos}@gmail.com
{felipe.dantas, robinson.alves}@ifrn.edu.br

Abstract. *The expansion of the Internet has grown the possibilities for fraudulent actions. Among these possibilities, we highlight the phishing activity, created with the objective of capturing the user's credentials through a false page similar to the original one. This work proposes PhishKiller, a tool capable of detecting and mitigating phishing attacks, using a proxy to intercept user-accessed addresses and featureless deep learning techniques to classify URLs. PhishKiller has an accuracy of 98.3% in detecting malicious addresses and an average processing time of 81.68ms per request.*

Resumo. *A expansão da Internet expandiu as possibilidades de ações fraudulentas. Entre essas ações, destaca-se a atividade de phishing, criada com o objetivo de capturar as credenciais do usuário através de uma página falsa similar à do serviço original. Diante disso, este trabalho propõe o PhishKiller, uma ferramenta capaz de detectar e mitigar ataques de phishing, utilizando um proxy para interceptação dos endereços acessados pelo usuário e técnicas de deep learning não baseadas em parâmetros para classificação dos URLs. As avaliações demonstraram a viabilidade do PhishKiller, que teve uma acurácia de 98.3% na detecção de endereços maliciosos e tempo de processamento médio de 81.68ms por requisição.*

1. Introdução

É inegável que a expansão da Internet acarretou inúmeras facilidades para a humanidade, como a troca de informações em tempo real, compras e transações bancárias online. Entretanto, ela também expandiu as possibilidades de ações fraudulentas graças à possibilidade do anonimato ao realizar tais ações. Dentre os possíveis ataques, se destaca a prática de phishing, que consiste em clonar uma página e utilizar técnicas de engenharia social para convencer um usuário desapercebido a inserir suas credenciais na página maliciosa. Tal prática tem se popularizado nos últimos anos, uma vez que não se faz necessário ao atacante usar métodos de quebra de *hashes* ou outras técnicas de exploração de vulnerabilidades, tais como *SQL injection* ou *session fixation*. A Figura 1 apresenta os passos necessários para um ataque de phishing bem sucedido.

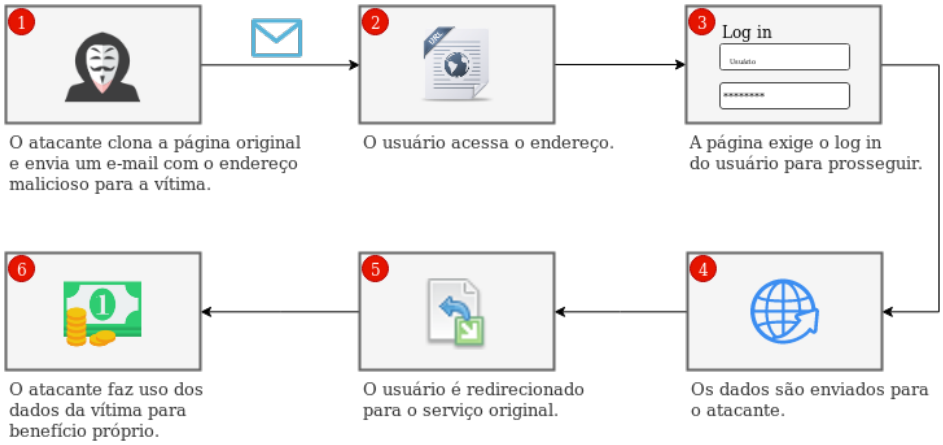

Figura 1. Sequência de passos para um ataque de phishing bem sucedido

Segundo o *Verizon Data Breach Investigation Report* (DBIR) 2017 [Team 2017], um relatório escrito através da colaboração de diversas organizações e fornecedores de soluções de segurança, 93% dos ataques de engenharia social tinham relação com phishing; 28% dos ataques de phishing são direcionados a usuários específicos (como gerentes ou pessoas com cargos elevados) e 66% dos malwares foram instalados através de anexos em e-mails. Constata-se que, embora seja uma técnica relativamente antiga, a prática de phishing continua sendo uma real ameaça à integridade dos dados de indivíduos e organizações.

[Volkamer et al. 2017] destaca que os ataques de phishing são bem sucedidos pelas seguintes falhas humanas: (i) falta de conhecimento em relação ao endereço acessado; (ii) incapacidade de identificar em qual URL confiar (no caso de e-mails); (iii) falta de acesso ao endereço real (por conta de redirecionadores ou URLs ofuscados); (iv) falta de consulta ao URL antes de clicar (acidentalmente ou por efeitos de hábitos); (v) incapacidade de distinguir URLs autênticos e de phishing.

Objetivando confrontar ameças de phishing, foram desenvolvidas técnicas para detecção de tais ataques. O combate ao phishing atualmente é feito, na maioria das vezes, de forma comunitária; através de sites públicos como o PhishTank[1] ou Google Safe Browsing[2], onde os usuários podem cadastrar endereços suspeitos para posterior análise e inserção nas *blacklists*. Uma limitação desse método é que todos os dias novas páginas maliciosas são criadas e se torna uma tarefa difícil catalogar novos URLs.

Outras abordagens comumente utilizadas são baseadas em processamento de linguagem natural [Sahingoz et al. 2019], parâmetros do domínio em questão [Desai et al. 2017] e análise do conteúdo da página [Jain and Gupta 2018]. Vale notar que essas técnicas podem não alcançar uma boa generalização para detecção de novos endereços na Internet. Além disso, as soluções consideradas estado da arte se preocuparam apenas em desenvolver mecanismos para detecção de URLs maliciosos, e não uma ferramenta no lado do cliente para efetivamente mitigar as ameaças.

Diante disso, este trabalho propõe o PhishKiller, um mecanismo capaz de detectar

[1]https://www.phishtank.com
[2]https://safebrowsing.google.com

ataques de phishing através de técnicas de *deep learning* não baseadas em parâmetros [KP et al. 2018] mediante uma abordagem não supervisionada treinada sobre um banco de dados abrangendo milhares de exemplos de URLs maliciosos e autênticos. A escolha de tal técnica se dá por conta da alta generalização obtida através de uma abordagem não orientada a parâmetros comuns do URL, que tendem a se parecer cada vez mais com os do endereço original. Desse modo, a abordagem é capaz de obter um alto grau de generalização. Ademais, o mecanismo proposto é incorporado em um proxy multiplataforma que opera no dispositivo do usuário, possibilitando o bloqueio de URLs suspeitos em tempo de execução. Desse modo, não é necessário que o usuário tome alguma ação para verificar se uma página é maliciosa, além de ser uma abordagem menos invasiva à privacidade do mesmo, já que nenhum dado será trafegado publicamente.

O restante deste trabalho está organizado da seguinte maneira: a Seção 2 apresenta a fundamentação teórica, com o intuito de familiarizar o leitor com alguns dos termos utilizados no trabalho; a Seção 3 aponta os trabalhos relacionados, com o objetivo de evidenciar as contribuições desta proposta; a Seção 4 descreve a detalhadamente a arquitetura do sistema proposto; a Seção 5 expõe os testes e resultados da arquitetura proposta; a Seção 6 apresenta as considerações finais e sugestões para trabalhos futuros.

2. Fundamentação Teórica

O uso de Redes Neurais e *Deep Learning* tem crescido a cada ano em diversos setores da indústria. Essas técnicas nos permitem predizer, com boa precisão, a probabilidade de algo ocorrer baseado em um treinamento anteriormente realizado. Com o intuito de situar o leitor na presente proposta, algumas técnicas e conceitos utilizadas neste trabalho são destacados.

No contexto de redes neurais, época (em inglês, *epoch*) é no número de vezes que o *dataset* é percorrido durante o treinamento. É importante não exagerar no número de épocas durante o treinamento, uma vez que o excesso de treinamento pode gerar problemas de *overfitting* [Srivastava et al. 2014], isto é, deixar o modelo viciado nos dados contidos no *dataset*. O conceito de época é muito relacionado e confundido com o conceito de *batch size*, que é a quantidade de dados passados para treinamento de uma única vez (uma vez que passar o *dataset* completo exigiria uma quantidade imensa de memória RAM). Normalmente são utilizados múltiplos de 2 como valores de *batch*.

O conceito de convolução é simplesmente a aplicação de um filtro a uma entrada que resulta em uma ativação. A aplicação repetida desse filtro em uma entrada resulta em um mapa de ativações, também chamado de mapa de recursos (do inglês, *feature map*), que indica os locais e a intensidade de um parâmetro detectado em uma entrada [Dumoulin and Visin 2016]. Uma variação das redes neurais é a LSTM (ou *Long short-term memory*), essas redes são caracterizadas por coibir a propagação do erro. Na prática, quanto menor o erro durante o treinamento, melhor sua acurácia nas predições [Hochreiter and Schmidhuber 1997].

[Mikolov et al. 2013] introduz Word2vec como um grupo de modelos utilizados para aprendizado de vetores de representação de palavras. Desse modo, é possível relacionar semanticamente palavras em um dado contexto. Nesse método, palavras semanticamente semelhantes estão mais próximas em um espaço vetorial. Uma das arquiteturas definidas pelo modelo Word2vec é a *Continuous Bag-of-Words* (CBOW), com essa

arquitetura o modelo é capaz de predizer uma palavra tendo base um conjunto de palavras da sua vizinhança. Além disso, esse método também é mais rápido e tem melhores representações para palavras frequentes.

3. Trabalhos Relacionados

Muitos esforços foram e vêm sendo desenvolvidos no meio acadêmico para detecção de ataques de phishing. Entretanto, poucos se preocuparam em também desenvolver soluções *client-side* para impedir que o ataque seja efetivo (ou seja, fazem apenas a classificação do URL). Nesta seção, são destacados os trabalhos que possuem maior similaridade com a presente proposta.

O trabalho de [Sahingoz et al. 2019] apresenta uma solução baseada em processamento de linguagem natural. Os autores também implementam 7 algoritmos de classificação com o objetivo de avaliar a acurácia da ferrameta em cada um, onde o melhor resultado foi com o algoritmo *Random Forest* [Breiman 2001], obtendo uma acurácia de 97.98%. O *dataset* utilizado pelos autores é composto de 73.575 endereços, dos quais 36.400 são de sites benignos e 37.175 de phishing. Outros pontos destacados são: independência do idioma e serviços de terceiros (como consultas WHOIS). Entretanto, os autores não propõem um mecanismo para efetivamente mitigar o ataque.

Já no trabalho de [Jain and Gupta 2018], os autores propõem uma solução voltada para o cliente que também independe de serviços de terceiros. A solução faz uso de parâmetros extraídos do URL e do código fonte da página, o que pode ser um problema em alguns casos, uma vez que a página maliciosa tende a ser idêntica à original (já que atacantes dispõem de ferramentas como o *Social Engineering Toolkit*[3] para clonar páginas com poucas alterações). O *dataset* utilizado pelos autores é composto de 2.141 URLs maliciosos e 1.918 legítimos. A acurácia do sistema alcançou uma acurácia geral de 99.09% utilizando o algoritmo *Random Forest*. Apesar de os autores destacarem o uso no lado do cliente, nenhum mecanismo para mitigação foi proposto.

A proposta de [Tyagi et al. 2018] avalia a performance de diferentes algoritmos de *machine learning* para detecção de endereços maliciosos. Os autores fazem uso de 30 parâmetros para classificação, onde alguns dependem de requisições à Internet para avaliação, como, por exemplo, consultas ao *favicon* da página, idade do domínio e indexação no Google. A melhor acurácia encontrada foi utilizando o algoritmo *Random Forest* e aplicando *Principal Component Analysis* (PCA) [Abdi and Williams 2010], no qual a taxa de acertos foi de 98.4%.

O trabalho de [Niakanlahiji et al. 2018] apresenta o PhishMon, uma ferramenta capaz de identificar endereços maliciosos sem uso de buscadores ou WHOIS. Entretanto, o serviço não deixa de fazer requisições externas, uma vez que alguns parâmetros são extraídos do serviço pertencente ao URL a ser analisado. O *dataset* utilizado pelos autores foi composto de 4.807 endereços maliciosos e 17.508 endereços legítimos. O mecanismo proposto teve uma acurácia de 95.4% utilizando o algoritmo *Random Forest*.

Já a proposta de [Desai et al. 2017] apresenta uma extensão de navegador para detecção de phishing utilizando *machine learning*. Os autores utilizaram um *dataset*

[3]https://github.com/trustedsec/social-engineer-toolkit

público da Universidade da Califórnia em Irvine (UCI)[4] composto por 11.055 URLs classificadas como phishing ou não, onde cada endereço tem 30 parâmetros para efeito de classificação. Essa solução depende de recursos de terceiros (como consultas ao índice de sites do Google), o que pode causar lentidão no processamento. Os autores também fizeram comparações da ferramenta com diferentes algoritmos, constatando que o *Random Forest* obteve a melhor acurácia: 96.11%. A estratégia utilizada pelos autores pode ser incômoda para o usuário, que tem que verificar se uma página é maliciosa a cada novo acesso. Além disso, o tempo de processamento pode aumentar drasticamente em caso de instabilidade do serviço utilizado para consulta.

O trabalho de [Marchal et al. 2014] propõe uma solução chamada PhishStorm. Tal ferramenta é capaz de classificar um endereço como maligno ou não através de regressão linear. Para isso, os autores fazem uso dos serviços oferecidos pelo Google Trends [5] para extrair dados sobre o endereço a ser analisado. O *dataset* utilizado pelos autores teve 53.089 endereços maliciosos e 48.009 endereços legítimos. O mecanismo proposto teve uma acurácia de 94.91%. Além do aumento do tempo de processamento ao se depender de serviços como o utilizado, os autores também não desenvolveram um mecanismo para mitigação dos ataques, deixando a criação de uma extensão para algum navegador como trabalho futuro.

Por fim, a proposta de [Belabed et al. 2012] expõe uma solução que combina a abordagem de *whitelist* com técnicas de *machine learning*. Tal mecanismo funciona como segue: se um endereço não estiver listado na *whitelist* é feita uma verificação de similaridade com os endereços cadastrados, caso a semelhança esteja dentro do limite, o endereço é legítimo; caso contrário, possívelmente é uma tentativa de phishing. Além disso, caso o URL seja extremamente diferente dos padrões cadastrados, a ferramenta faz uso do algoritmo *Support Vector Machine* (SVM) [Scholkopf and Smola 2001] para classificação do endereço. Destaca-se aqui a dependência de buscadores para avaliação da legitimidade do site. O *dataset* utilizado pelos autores consistiu em 400 páginas (metade phishing e metade legítima) e testado contra 450 páginas (200 legítimas e 250 phishing). O modelo de classificação dos autores alcançou uma acurácia de 98.4%. Entretanto, a utilização de uma lista para URLs legítimos pode ser um incômodo para o usuário, que deve inserir os sites que acessa para que não aconteça o bloqueio deles.

A Tabela 1 apresenta um comparativo entre os trabalhos selecionados levando em consideração itens de extrema importância para uma solução robusta de combate a ataques de phishing. A lista abaixo apresenta a legenda dos itens comparados:

1. **Independência de serviços de terceiros:** a solução não faz consultas a servidores WHOIS ou indexadores de sites;
2. **Faz mitigação:** propõe alguma ferramenta para efetivamente mitigar o ataque;
3. **Acurácia:** porcentagem de acertos do mecanismo avaliado.

Como pode ser notado na Tabela 1, foi possível obter uma boa taxa de acertos em todas as soluções propostas. Entretanto, poucos trabalhos se preocuparam em desenvolver um mecanismo que não depende de consultas a outras bases de dados na Internet e, simultaneamente, capaz de mitigar os ataques no lado do cliente. O presente trabalho tem

[4]https://archive.ics.uci.edu/ml/datasets/phishing+websites

[5]https://trends.google.com/trends

como objetivo de preencher essas lacunas, sendo uma ferramenta de fácil implantação e alta performance.

Proposta	#1	#2	#3
[Sahingoz et al. 2019]	✓		97.98%
[Jain and Gupta 2018]	✓		99.09%
[Tyagi et al. 2018]			98.40%
[Niakanlahiji et al. 2018]	✓		95.40%
[Desai et al. 2017]		✓	96.11%
[Marchal et al. 2014]			94.91%
[Belabed et al. 2012]			98.40%
PhishKiller	✓	✓	98.30%

Tabela 1. Tabela comparativa entre as publicações relacionadas

4. Arquitetura do PhishKiller

A ferramenta aqui proposta tem como objetivos principais detectar e mitigar tentativas de phishing no lado do cliente de forma rápida, com alta precisão e de forma a não violar a privacidade dos utilizadores. Para isso, a solução faz uso de um proxy responsável por capturar o tráfego gerado pelo navegador utilizado e fazer o encaminhamento para uma API local com o objetivo de verificar se o endereço que está sendo acessado no momento é uma possível tentativa de phishing. A Figura 2 apresenta detalhes da arquitetura do sistema.

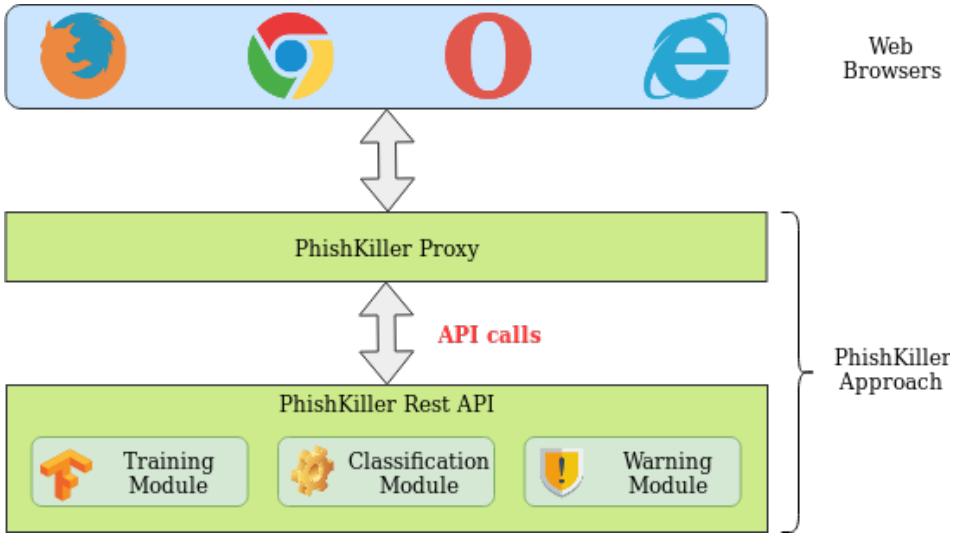

Figura 2. Arquitetura do PhishKiller

Como se pode observar na Figura 2, a arquitetura do PhishKiller é composta por dois módulos principais: o proxy e o mecanismo em si. Uma vez configurado no navegador do cliente, o proxy é responsável por interceptar as requisições feitas pelo usuário, extrair os URLs e enviá-los para o método de classificação da API. Esse processo é feito com o auxílio das funções do *framework* web Tornado[6]. Para evitar sobrecarga, apenas

[6]https://www.tornadoweb.org

a primeira requisição feita é analisada. Além disso, após constatar que uma página é benigna, seu endereço é adicionado a uma *whitelist* em um banco de dados SQLite para evitar futuras predições desnecessárias. Caso um endereço seja uma possível tentativa de phishing, a API retorna uma *flag* ao proxy, que por sua vez trata de encaminhar o usuário à uma página contendo o endereço, a probabilidade de ser uma tentativa de phishing e um aviso sobre os riscos.

O processo de treinamento do modelo segue os seguintes passos: (i) representação dos caracteres em palavras; (ii) processamento dos dados com Word2Vec; (iii) processo de convolução com LSTM. O primeiro passo consiste em converter os caracteres para inteiros únicos que o representem para uso nos próximos passos, além disso, foi definido um tamanho padrão de endereço de 100 caracteres; desse modo, são adicionados zeros para completar o URL caso o mesmo não tenha esse tamanho. O segundo passo é a incorporação com Word2vec utilizando seu modelo CBOW, para que as características de cada URL sejam utilizadas no treinamento (iii). O terceiro e último passo consiste em treinar o modelo através de convoluções 1D e LSTM com o auxílio das bibliotecas TensorFlow[7] e Keras[8].

A vantagem dessa abordagem é não requisitar nenhuma ação do usuário para verificação. Vale notar, também, que a privacidade do usuário não é comprometida, uma vez que não é necessário realizar nenhuma consulta a terceiros sobre o endereço acessado. Além disso, destacamos a possibilidade de utilização do proxy em qualquer sistema operacional com suporte à linguagem Python e em qualquer navegador que tenha opções para configuração de proxy.

4.1. *Dataset* utilizado

O *dataset* utilizado para treinamento da rede neural foi construído a partir da coleta de dados (em inglês, *scraping*) de fontes públicas. Para sites maliciosos foram extraídos dados do PhishTank[9], OpenPhish[10], Phishbank[11] e Malware Domain List[12]. Para sites legítimos, utilizamos parte da lista de sites indexados pelo Alexa[13]. No total, foram utilizados 452.835 endereços, dos quais 108.013 são de phishing e 344.829 de sites benignos. A quantidade de dados utilizados para treinamento foi de 80% do tamanho total do *dataset*, enquanto 20% foi destinado para validação da rede treinada. Sua estrutura segue o padrão CSV definido pela RFC 4180 [Shafranovich 2005], onde são utilizados URL e status como rótulos. O número 1 é utilizado para representar um endereço malicioso e o 0 para um endereço benigno.

5. Avaliação e Resultados

A avaliação do sistema proposto foi feita em um computador com processador Intel Core i7-6500U de 2.40GHz, 8GB de memória RAM e Internet com velocidade de 30Mbps. O sistema operacional utilizado para os testes foi a distribuição GNU/Linux Ubuntu 18.04.2

[7]https://www.tensorflow.org
[8]https://keras.io
[9]https://www.phishtank.com
[10]https://openphish.com
[11]https://phishbank.org/
[12]https://www.malwaredomainlist.com
[13]https://www.alexa.com/topsites

Épocas	Tempo *batch size* = 16	Tempo *batch size* = 32
5	38.96	23.58
10	69.13	46.70
15	103.53	69.51
20	139.96	95.12
25	175.83	116.86
30	205.54	142.44
100	884.32	440.57

Tabela 2. Comparação entre o tempo necessário para diversos treinamentos

LTS de 64-bits e a ferramenta *curl 7.64.0*[14] foi utilizada para fazer requisições via proxy. Para treinamento, foi utilizado um computador com processador Intel Xeon E5-2620 de 2.40GHz e 8 núcleos, 16GB de memória RAM e mesmo sistema operacional.

Foram realizados 7 treinamentos, com o objetivo de encontrar o melhor número de épocas para se treinar o modelo. Além disso, é feita uma comparação para dois valores de *batch size*: 16 e 32. Como pode ser observado na Figura 3, mesmo com um baixo número de épocas, a acurácia do sistema ao ser testado na porção de URLs do *dataset* destinados à validação se mostrou aceitável. A Tabela 2 apresenta a relação do tempo (em minutos) necessário para treinamento conforme a variação de épocas e *batch sizes*.

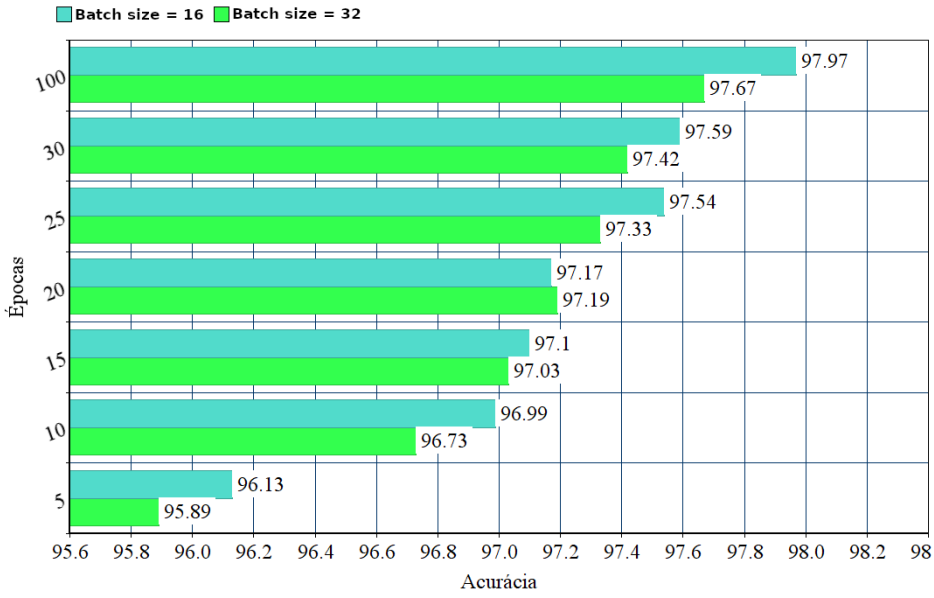

Figura 3. Comparação entre número de épocas, *batch size* e acurácia

Para evitar problemas de *overfitting*, o modelo utilizado para os próximos testes foi o obtido ao se realizar o treinamento com 30 épocas e *batch size* de 16, que obteve acurácia de 97.59% para os 20% de dados do *dataset* utilizados para avaliação. Para validação do mecanismo proposto, foram feitos testes automatizados na API com 2000 URLs (1000 malignos e 1000 benignos) externos aos testados no processo de treinamento, obtendo

[14]https://curl.haxx.se/

uma acurácia de 98.3%. Para se realizar esses testes, foram feitas requisições com o *curl* devidamente configurado para redirecionar o tráfego para o proxy do PhishKiller.

Em relação ao tempo de processamento da requisição, foi possível notar que, nos mesmos testes, houve pouca alteração ao processo normal (sem a utilização de um proxy): em média, 81.68ms a mais para completar a requisição. Logo, demonstramos que é possível a utilização do sistema para uma navegação mais segura em troca de frações de milissegundos para processamento do endereço acessado.

6. Conclusão

Neste trabalho apresentamos uma ferramenta capaz de detectar e mitigar ataques de phishing. Para isso, a ferramenta intercepta as requisições feitas pelo usuário através de um proxy multiplataforma e compatível com diversos navegadores, que é responsável por consultar uma API Rest para classificar o URL interceptado através de métodos de *deep learning* e bloquear as páginas maliciosas encontradas. Além disso, destacamos a alta acurácia do sistema e baixo tempo de processamento, tornando seu uso viável em cenários reais.

Por utilizar uma abordagem não baseada em parâmetros comuns para classificação dos endereços, o processo de treinamento exige uma quantidade imensa de dados, o que pode fazê-lo menos preciso no caso de um *dataset* pouco elaborado. Porém, com uma base de dados consistente, o sistema apresenta uma boa resiliência em relação a novos tipos de ataques de phishing, que tendem a se parecer a cada vez mais com os serviços originais. Os experimentos mostraram também o alto tempo gasto para treinamento dos modelos, que pode variar de acordo com o *dataset*, quantidade de épocas e *batch size*.

Como trabalho futuro, espera-se adaptar o sistema para as novas tecnologias de redes de computadores, como o paradigma de Redes Definidas por Software (SDN), no qual um controlador centralizado tem domínio de toda a rede. Nesse cenário, esse sistema pode ser implementado como um módulo no controlador para analisar e mitigar qualquer acesso a páginas maliciosas dentro da rede. Além disso, é possível disponibilizar a API Rest publicamente para uso de desenvolvedores ou outros interessados. Esperamos também que este trabalho incentive novas pesquisas na área de mitigação de ataques de phishing e que possa contribuir para conscientização dos usuários da Internet sobre as ameaças que estamos sujeitos diariamente ao acessar a rede mundial de computadores.

7. Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e ao Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN) pelo apoio ao desenvolvimento dessa pesquisa. Agradecemos também à Actions Security pela infraestrutura cedida para a realização dos treinamentos.

Referências

Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.

Belabed, A., Aïmeur, E., and Chikh, A. (2012). A personalized whitelist approach for phishing webpage detection. In *2012 Seventh International Conference on Availability, Reliability and Security*, pages 249–254. IEEE.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Desai, A., Jatakia, J., Naik, R., and Raul, N. (2017). Malicious web content detection using machine leaning. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1432–1436. IEEE.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Jain, A. K. and Gupta, B. B. (2018). Towards detection of phishing websites on client-side using machine learning based approach. *Telecommunication Systems*, 68(4):687–700.

KP, S. et al. (2018). A short review on applications of deep learning for cyber security. *arXiv preprint arXiv:1812.06292*.

Marchal, S., François, J., State, R., and Engel, T. (2014). Phishstorm: Detecting phishing with streaming analytics. *IEEE Transactions on Network and Service Management*, 11(4):458–471.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Niakanlahiji, A., Chu, B.-T., and Al-Shaer, E. (2018). Phishmon: A machine learning framework for detecting phishing webpages. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 220–225. IEEE.

Sahingoz, O. K., Buber, E., Demir, O., and Diri, B. (2019). Machine learning based phishing detection from urls. *Expert Systems with Applications*, 117:345–357.

Scholkopf, B. and Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Shafranovich, Y. (2005). Common format and mime type for comma-separated values (csv) files. RFC 4180, RFC Editor.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Team, V. R. (2017). 2017 data breach investigations report.

Tyagi, I., Shad, J., Sharma, S., Gaur, S., and Kaur, G. (2018). A novel machine learning approach to detect phishing websites. In *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 425–430. IEEE.

Volkamer, M., Renaud, K., Reinheimer, B., and Kunz, A. (2017). User experiences of torpedo: tooltip-powered phishing email detection. *Computers & Security*, 71:100–113.