# Estudo comparativo do módulo rastreador de *scanners* de vulnerabilidade web de código aberto

**Danilo Pereira Escudero**[1], **Ewerton R. Andrade**[2], **Routo Terada**[1]

[1] Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP)
São Paulo/SP – Brasil.

[2]Grupo de Pesquisa em Segurança, Algoritmos, Computação e Inovação (SACI)
Departamento Acadêmico de Ciência da Computação (DACC)
Fundação Universidade Federal de Rondônia (UNIR)
Porto Velho/RO – Brasil.

dpe@ime.usp.br, ewerton.andrade@unir.br, rt@ime.usp.br

***Resumo.*** *Scanners de vulnerabilidades web auxiliam na detecção de vulnerabilidades em sistemas web de forma automatizada. Um scanner de vulnerabilidade web é dividido em três módulos: módulo rastreador, módulo atacante e módulo analisador. O módulo rastreador é um dos principais limitadores da eficácia apresentado pelos scanners, pois, se a ferramenta não é capaz de acessar todas as funcionalidades de um sistema web, muitas páginas vulneráveis não serão testadas. Este trabalho apresenta um estudo comparativo do módulo rastreador de scanners de código aberto, sugerindo adaptações para que sejam atingidos melhores resultados.*

## 1. Introdução

Sistemas *Web* são utilizados em larga escala, porém não existe sistema *Web* "totalmente seguro". Assim sendo, informações confidenciais são expostas a todo tempo [Tunggal 2020].

Para diminuir os custos com verificações e dar mais agilidade na execução de testes de segurança, existem ferramentas que auxiliam na detecção de vulnerabilidades em sistemas *Web*. Essas ferramentas são chamadas de *scanners* e são utilizadas para realizar testes de segurança automatizados; usualmente, sem informações ou acessos privilegiados. Genericamente, um *scanner* de vulnerabilidade *web* é dividido em três módulos: módulo rastreador (*crawler*), módulo atacante (*attacker*) e módulo analisador (*analysis*). Após suas análises, gera-se um relatório de segurança em alto nível que pode ser utilizado pela equipe de tecnologia da informação para auxiliar nas atualizações necessárias a fim de contornar possíveis falhas de segurança [Idrissi et al. 2017].

Todavia, tais ferramentas ignoram diversos tipos de vulnerabilidades e apresentam resultados com muitos falsos positivos, falsos negativos e, principalmente, baixa capacidade de rastreabilidade [Alsaleh et al. 2017, Salas and Martins 2015]. Estudos recentes apontam que a principal limitação dos *scanners* atuais é a capacidade de indexação dos *crawlers* [Deepa et al. 2018, Alsaleh et al. 2017]. O módulo rastreador (*crawler*) de um *scanner* de vulnerabilidades *web* recupera as páginas correspondentes, segue os links e redireciona para identificar todas as páginas acessíveis no aplicativo a partir de uma URL de

entrada. Além disso, o rastreador identifica todos os pontos de entrada do aplicativo, tais como formulários e *upload* de arquivos [Doupé et al. 2010]. Assim, utilizar um *scanner* com um *crawler* eficiente, com uma boa capacidade de indexação de páginas, formulários e demais pontos de entrada é essencial para uma boa análise de segurança automatizada.

### 1.1. Objetivo deste trabalho

Este trabalho tem por objetivo principal apresentar um estudo comparativo do módulo rastreador de *scanners* de código aberto, principalmente quanto a sua capacidade de indexação de páginas e arquivos do sistema em teste. Complementarmente, sempre que possível, almeja-se apontar melhorias para as soluções analisadas. Vale destacar que, ao longo deste trabalho, adotou-se *crawler* como módulo rastreador.

## 2. Método e descrição dos testes

O método adotado durante o desenvolvimento deste estudo foi o da pesquisa experimental exploratória [Wazlawick 2017], uma vez que os autores não tinham noção de todas as peculiaridades das ferramentas analisadas. Ao entrarem em contato com os sistemas analisados, os autores desenvolveram experimentos e coletaram informações, comuns a todos os *scanners*, envolvendo métricas relevantes para o contexto de testes automatizados.

Para facilitar a replicabilidade dos testes e permitir que outros pesquisadores validem os resultados expostos na Seção 3, utilizou-se virtualização. Como cada um dos *scanners* testado possui múltiplas dependências, foi necessário adicionar repositórios extras e instalar bibliotecas que muitas vezes são desatualizadas ou descontinuadas. Nesse sentido, para facilitar a reprodução do cenário de testes, foi realizada a instalação e configuração de todas as ferramentas e suas dependências e, posteriormente, foi gerada uma imagem da máquina. Vale destacar que essa imagem está disponível em: `https://bit.ly/2TWWCbN`, e os manuais e *scripts* para realização dos testes localizam-se na pasta `~/testes` do disco virtual da máquina.

Além disso, para que haja parâmetro de comparação, ressalta-se que as máquinas utilizadas durante os testes possuíam as seguintes especificações:

- **Máquina física:** processador Inte Core i7-8750H, com 6 núcleos físicos e 12 *threads*, memória RAM 16 GB DDR4, modelo de notebook Dell G7-7588, sistema operacional Linux Ubuntu 20.04 LTS, software de virtualialização VirtualBox 6.1.10_Ubuntu r138449, com máquinas instanciadas no disco SSD com 560mbs de leitura e 530mbs de escrita;
- **Máquina virtual:** 6 núcleos de processadores virtuais, 4 GB de memória RAM dedicada, sistema operacional Kali Linux 2020.2 64-Bit e disco VDI de 25 GB.

Destaca-se que a escolha do sistema operacional Kali Linux para a máquina virtual se deu pela sua popularidade dentro da comunidade de testes de segurança automatizados, bem como por nativamente possuir diversas bibliotecas e repositórios exigidos pelos *scanners* testados.

No tocante aos *scanners* testados neste trabalho, optou-se por utilizar somente aqueles com código aberto, boa documentação, e relativa popularidade na comunidade técnico-científica. A categoria de ferramentas selecionadas foi a de *scanners* não desenvolvidos para um único tipo de vulnerabilidade, mas sim uma ferramenta mais abrangente com um *crawler* próprio e capaz de testar diversas vulnerabilidades.

Após uma pesquisa na literatura, foram encontradas onze ferramentas com essas características. São elas: w3af, Grendel-Scan, Paros, OpenVAS, Arachni, Vega, soapUI, Wapiti, Ironwasp, SkipFish e Nikto. Algumas dessas ferramentas não foram incluídas no experimento por terem desempenho insatisfatório em estudos recentes de outros autores. Por exemplo, as ferramentas soapUI e Arachni apresentam alta porcentagem de falsos positivos e falsos negativos [[Mburano and Si 2018], [Mburano and Si 2018]]. Dessa forma, os testes deste trabalho restringiram-se ao módulo rastreador das seguintes ferramentas:

- Wapiti – 3.0.3 (`https://wapiti.sourceforge.io`)
- Paros – 3.2.13 (`http://www.parosproxy.org/`)
- w3af – 2019.1.2 (`http://w3af.org/`)
- Subgraph Vega – 1.0 (`https://subgraph.com/vega`)
- Nikto – 2.1.6 (`https://cirt.net/Nikto2`)
- SkipFish – 2.10b (`https://tools.kali.org/web-applications/skipfish`)

Apesar dessas ferramentas permitirem configurações personalizadas e avançadas (*e.g.*, utilização de *cookies*, fornecimento de credenciais de autenticação, definição de número de requisições, entre outras), é importante salientar que não foi realizada nenhuma parametrização dos *scanners* e seus módulos de rasteamento. Julgou-se pertinente avaliar apenas a capacidade de cada *scanner* formular sua estratégia de ataque automatizado, ou seja, como cada um irá criar usuários, autenticar explorando vulnerabilidades das páginas de *login*, autenticar com usuário potencialmente criado pela ferramenta, ou, ainda, verificar a capacidade de realizar *upload* de arquivos.

Em relação às configurações e características das ferramentas, foi definida a URL de ataque 127.0.0.1 para todos os *scanners*. O Subgraph Vega e o Paros apresentam o relatório em sua própria interface gráfica e também possuem um *proxy* próprio. A ferramenta Skipfish também possui *proxy* próprio, porém, o relatório foi configurado para gerar um arquivo HTML. Para os demais *scanners*, foi utilizado como *proxy* o Burp Suite Community 2020.4.1 para a realização de todos os testes, e solicitado relatório em HTML. As demais configurações padrão de cada *scanner* foram mantidas.

É importante observar que, como em qualquer experimento desta natureza, os resultados podem divergir dependendo das especificações do cenário de testes (*i.e.*, versão do *software*, sistema operacional, *hardware* etc.). Destaca-se que durante os experimentos deste trabalho foram realizadas as repetições necessárias até que o desvio padrão dos resultados dos testes fosse menor que 1%.

## 2.1. Métricas Utilizadas

As principais métricas apresentadas por trabalhos anteriores são a quantidade de falsos positivos e a quantidade de falsos negativos nas varreduras dos *scanners* [Alsaleh et al. 2017, Dalalana Bertoglio and Zorzo 2017]. Apesar de alguns trabalhos medirem a cobertura dos *crawlers*, nenhum discute isoladamente as métricas de rastreamento obtidas. Isso motivou esta pesquisa a criar as seguintes métricas específicas para o módulo rastreador dos *scanners*: recursos indexados, recursos acessados, páginas vulneráveis exploradas, interação entre os módulos, gera páginas infinitamente, realiza *upload* durante ataque e realiza ataque baseado em dicionário.

A métrica *Recursos indexados* significa a quantidade de recursos que foram indexados pela ferramenta, podendo ser páginas *web*, diretórios ou outros tipos de arquivos. Porém, ser capaz de indexar não significa que a ferramenta consegue obter acesso, mas apenas sabe que o recurso existe dentro do servidor. Já para os *recursos acessados*, o *scanner* é capaz de indexar e acessar o recurso, enquanto que *páginas vulneráveis exploradas* é a quantidade de páginas em que a ferramenta é capaz de realizar requisições válidas e testar se há vulnerabilidades. A métrica *interação entre os módulos*, por sua vez, é a capacidade do *scanner* retornar ao módulo rastreador após ter realizado um ataque e descoberto uma vulnerabilidade, fazendo novas descobertas com base na vulnerabilidade encontrada. Já a métrica *gera páginas infinitamente* significa que a ferramenta não sabe identificar páginas dinâmicas e entra em *loop*. As páginas geradas a partir de calendários, por exemplo, podem fazer uma ferramenta mal programada entrar em um *loop* infinito. Por sua vez, *realiza upload durante ataque* avalia se a ferramenta é capaz de realizar *upload* de arquivos durante a execução da varredura, enquanto que *realiza testes repetitivos* verifica se a ferramenta gera requisições repetidas para uma mesma URL, especialmente quando há páginas que possuem *hiperlinks* em várias partes do sistema. Se a ferramenta não verificar se a página já foi testada, pode gerar testes repetitivos. Por último, *realiza ataque baseado em dicionário* mede se a ferramenta utilizou algum dicionário de usuários e senhas para tentar autenticar nas páginas de *login*.

Analisando todas as métricas descritas anteriormente, nota-se que é imprescindível a utilização de *proxy* para que seja possível verificar exatamente o tipo de requisição feita pelos *scanners*, quais os dados enviados na requisição e se foi bem sucedida. As métricas também permitem analisar a quantidade de falsos negativos nas varreduras por motivo de mal funcionamento do módulo rastreador.

## 3. Resultados dos testes

O sistema *web* usado para testar os *scanners* foi o WackoPicko [Doupé et al. 2019]. A escolha desse sistema se deu por ele ter todas suas vulnerabilidades documentadas e haver diversos estudos na literatura que utilizam esse sistema para também testar *scanners*, como em [Doupé et al. 2010, Li and Xue 2011].

O WackoPicko foi utilizado em *localhost* por meio de contêiner Docker disponibilizado pelos seus autores. Sobre esse aspecto, destaca-se que a utilização do sistema em *localhost* é importante, pois assim não há variação significativa na latência de rede. Caso não fosse feito dessa forma, os resultados poderiam ficar incoerentes. O contêiner e a máquina virtual possuem a configuração da placa de rede em modo *bridge*.

Salienta-se que esse sistema *web* apresenta 142 arquivos e diretórios. Desses arquivos, o sistema possui 15 páginas que possuem vulnerabilidades, logo, conseguir rastreá-las é necessário para diminuir o número de falsos negativos do *scanner*.

A tabela 1 apresenta os resultados de todos os *scanners* estudados para todas as métricas adotadas. Ressalta-se que o objetivo deste trabalho é apresentar um diagnóstico do módulo rastreador que pode ser usado para auxiliar na melhoria dos *scanners* em novas versões. Os resultados apresentados também podem ser utilizados para auxiliar pesquisas de outros autores. Não é possível classificar qual ferramenta é melhor, pois, para chegar a

[1]Mas gerou um grande número de páginas dinamicamente.

**Tabela 1. Visão geral dos resultados obtidos nos testes.**

| | Wapiti | Paros | w3af | Subgraph | Nikto | SkipFish |
|---|---|---|---|---|---|---|
| *Recursos indexados* | 36 | 54 | 27 | 123 | 13 | 77 |
| *Recursos acessados* | 36 | 50 | 27 | 102 | 13 | 77 |
| *Páginas Vulneráveis exploradas* | 10 | 7 | 6 | 7 | 1 | 4 |
| *Interação entre módulos* | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| *Gera páginas infinitamente* | ✗[1] | ✗ | ✗ | ✗ | ✗ | ✗ |
| *Realiza upload durante o ataque* | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| *Realiza testes repetitivos* | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ |
| *Realiza ataque baseado em dicionário* | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |

✓- Sim *(possui, gera, realiza ou testa)*; ✗- Não *(não possui, não gera, não realiza ou não testa)*.

essa conclusão, seria necessário expandir os experimentos realizados nesta pesquisa para vários sistemas *Web*, abrangendo diversas tecnologias diferentes.

## 4. Discussões

Conhecer as limitações de um *crawler* é importante para ponderar sobre a eficácia de um *scanner*, pois isso impacta principalmente na métrica de falsos negativos gerados pela ferramenta. Apesar deste trabalho não medir os falsos negativos dos *scanners*, mensurou-se a limitação que o *crawler* provoca, mostrando que diversas páginas com vulnerabilidades não foram indexadas.

Nos experimentos, foram ativados quase todos os módulos de *crawler* da ferramenta w3af, mas ela não conseguiu sequer indexar todas as páginas sem autenticação. Uma forma mais eficiente de se rastrear é utilizar bibliotecas padrão de processamento de páginas HTML para extrair *hiperlinks* em uma página da *web*. O rastreador procura os atributos (como href, src e action) no conteúdo HTML e eventos JavaScript (como window.open, window.location, .load, .location.assign, .href, .action e .src) para identificar as URLs existentes em uma página da *Web* [Deepa et al. 2018]. Apesar das ferramentas Subgraph Vega e Wapiti utilizarem essa técnica de seguir os *links*, o *scanner* Subgraph Vega não obteve bons resultados para páginas vulneráveis por não conseguir realizar requisições com autenticação. Já o *scanner* Wapiti conseguiu realizar requisições com autenticações bem sucedidas, mas não conseguiu atingir todos as URLs do sistema WackoPicko.

Sobre autenticação, é importante que o *crawler* dos *scanners* realize tentativas de *login* com base em um dicionário. A página de autenticação "/ADMIN/LOGIN.PHP", possui um usuário *admin* com senha *admin*, porém, apesar de ser uma abordagem comum, nenhuma ferramenta realizou tentativa de autenticação com usuários e senhas comuns (*e.g.*, admin / admin).

O Wapiti gerou 37 páginas baseadas em calendários, enquanto a média de outras ferramentas foi de 8 páginas. Se o *crawler* de uma ferramenta não prever essas páginas dinâmicas, o *scanner* pode entrar em um *loop* infinito.

Um outro aspecto relevante é que os mecanismos de pesquisa, tais como Google e Bing, dependem amplamente de robôs da *Web* para coletar informações da *Web*. Dessa forma as atividades de rastreamento podem ser reguladas do lado do servidor, implantando

o Protocolo de Exclusão de Robôs em um arquivo chamado "ROBOTS.TXT". Porém, os mecanismos de busca não são obrigados a seguir esse padrão, mas os robôs éticos (e muitos comerciais) seguirão as regras especificadas no ROBOTS.TXT [Sun et al. 2007, Mallawaarachchi et al. 2020]. Uma das técnicas para rastrear páginas *Web* de um sistema é analisar o arquivo ROBOTS.TXT. O *scanner* w3af foi o único a apresentar um *plugin crawler* capaz de analisar esse arquivo.

Exceto o Wapiti, todos os *scanners* geram mais de 50% de falsos negativos, considerando apenas as páginas com vulnerabilidades que não foram rastreados por cada ferramenta.

## 5. Considerações Finais

A principal contribuição deste trabalho foi a apresentação de um estudo comparativo do módulo rastreador de *scanners* de código aberto. Como resultado, foi possível perceber que é importante combinar diferentes técnicas de ataque (*e.g.*, robots.txt, seguir html e javascript, dicionário de diretórios e arquivos etc). Além disso, ficou evidente que não conseguir fazer autenticação no sistema sob ataque faz com que as ferramentas não alcancem diversas páginas vulneráveis, o que gera resultados insatisfatórios. Conclui-se que técnicas mais sofisticadas de ataques de autenticação (*e.g.*, ataques baseados em dicionários) poderiam melhorar significativamente tal aspecto.

Em relação aos *scanners* analisados, nenhum deles combinou todas as técnicas para rastrear vulnerabilidades de forma automatizada. O Wapiti foi a ferramenta que alcançou mais páginas vulneráveis, e o Subgraph Vega indexou mais recursos. Nota-se que os resultados apresentados pelos *scanners* não são eficazes. Por esse motivo, é necessário realizar testes manuais em sistemas *Web* que ainda entrarão em produção a fim de encontrar vulnerabilidades.

Portanto, novamente ficou evidente que ter um *crawler* eficiente é fundamental para que um *scanner* tenha resultados mais confiáveis/eficazes. Todavia, em virtude de alguns aspectos inerentes à tecnologia, acredita-se que este trabalho possa ser continuado por meio da expansão da análise comparativa dos *scanners*, bem como pelo desenvolvimento de extensões para as ferramentas com potencial de aprimoramento.

## Referências


Alsaleh, M., Alomar, N., Alshreef, M., Alarifi, A., and Al-Salman, A. (2017). Performance-based comparative assessment of open source web vulnerability scanners. *Security and Communication Networks*, 2017:1–14.

Dalalana Bertoglio, D. and Zorzo, A. F. (2017). Overview and open issues on penetration test. *Journal of the Brazilian Computer Society*.

Deepa, G., Thilagam, P. S., Khan, F. A., Praseed, A., Pais, A. R., and Palsetia, N. (2018). Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications. *International Journal of Information Security*, 17.

Doupé, A., Cova, M., and Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In *LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Doupé, A., Cova, M., and Vigna, G. (2019). Wackopicko. Last accessed 14 October 2019. https://github.com/adamdoupe/WackoPicko.

Idrissi, S., Berbiche, N., Guerouate, F., and Shibi, M. (2017). Performance evaluation of web application security scanners for prevention and protection against vulnerabilities. *International Journal of Applied Engineering Research*, 12(21):11068–11076.

Li, X. and Xue, Y. (2011). BLOCK: A Black-bOx approach for detection of state violation attacks towards web applications. In *ACM International Conf. Proceeding Series*.

Mallawaarachchi, V., Meegahapola, L., Madhushanka, R., Heshan, E., Meedeniya, D., and Jayarathna, S. (2020). Change Detection and Notification of Web Pages: A Survey. *ACM Computing Surveys (CSUR)*, 53(1):1–35.

Mburano, B. and Si, W. (2018). Evaluation of web vulnerability scanners based on owasp benchmark. In *2018 26th International Conference on Systems Engineering (ICSEng)*, pages 1–6.

Salas, M. I. P. and Martins, E. (2015). A Black-Box Approach to Detect Vulnerabilities in Web Services Using Penetration Testing. *IEEE Latin America Transactions*, 13(3).

Sun, Y., Zhuang, Z., and Giles, C. L. (2007). A large-scale study of robots.txt. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 1123–1124, New York, NY, USA. Association for Computing Machinery.

Tunggal, A. T. (2020). UpGuard – The 36 Biggest Data Breaches [Updated for 2020]. Last accessed 27 July 2020. https://www.upguard.com/blog/biggest-data-breaches.

Wazlawick, R. S. (2017). *Metodologia de pesquisa para ciência da computação*. Elsevier, 3 edition.