



**VITOR ORIEL DE CASTRO NUNES BORGES**

**FUZZINGTOOL: FERRAMENTA PARA TESTES DE  
INTRUSÃO EM APLICAÇÕES WEB**

**LAVRAS – MG**

**2022**

**VITOR ORIEL DE CASTRO NUNES BORGES**

**FUZZINGTOOL: FERRAMENTA PARA TESTES DE INTRUSÃO EM APLICAÇÕES  
WEB**

Trabalho de Conclusão de Curso apresentado  
à Universidade Federal de Lavras, como  
parte das exigências do Curso de Ciência da  
Computação, para a obtenção do título de  
Bacharel

Prof. Joaquim Quinteiro Uchôa  
Orientador

**LAVRAS – MG**  
**2022**

**Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos  
da Biblioteca Universitária da UFLA**

Borges, Vitor Oriel de Castro Nunes

FuzzingTool: Ferramenta para Testes de Intrusão em Aplicações Web / Joaquim Quinteiro Uchôa. 2<sup>a</sup> ed. rev., atual. e ampl. – Lavras : UFLA, 2022.

11 p. : il.

TCC(Graduação)–Universidade Federal de Lavras, 2022.

Orientador: Prof. Joaquim Quinteiro Uchôa.

Bibliografia.

1. Teste de Intrusão. 2. Aplicações Web. 3. Fuzzing. I. Uchôa, Joaquim Quinteiro. II. Título.

**VITOR ORIEL DE CASTRO NUNES BORGES**

**FUZZINGTOOL: FERRAMENTA PARA TESTES DE INTRUSÃO EM APLICAÇÕES  
WEB**

Trabalho de Conclusão de Curso apresentado  
à Universidade Federal de Lavras, como  
parte das exigências do Curso de Ciência da  
Computação, para a obtenção do título de  
Bacharel

APROVADA em 19 de abril, 2022.

Joaquim Quinteiro Uchôa      UFLA  
Luiz Henrique Andrade Correia   UFLA  
Victor Grudtner Boell

Prof. Joaquim Quinteiro Uchôa  
Orientador

**LAVRAS – MG  
2022**

*Dedico ao meu avô Vitor Borges dos Reis e à minha avó Mariana de Jesus dos Reis*

## **AGRADECIMENTOS**

Agradeço a minha família e a todos os meus amigos que me apoiaram e acreditaram no meu potencial durante todo esse período de graduação. Agradeço ao Victor Grudtner por ter me orientado e encaminhado à área de segurança da informação, e ao Joaquim Uchôa pela sua orientação.

## RESUMO

A cada dia as aplicações Web estão sendo usadas para atividades complexas, a fim de atender as demandas de mercado. Com o crescente uso e avanço de suas tecnologias, faz-se necessário uma maior preocupação quanto à segurança da informação. Para tal, este TCC apresenta uma ferramenta desenvolvida para testes de intrusão em aplicações Web, o FuzzingTool. A ferramenta faz uso da técnica de *fuzzing* para localizar falhas nessas aplicações, com resultados bastante promissores em testes, por exemplo, que permitiram identificar vários sites auxiliares, potencialmente inseguros, a partir de um domínio principal.

**Palavras-chave:** Teste de intrusão, Aplicações Web, Fuzzing

## SUMÁRIO

<b>1 INTRODUÇÃO</b> . . . . .	6
<b>REFERÊNCIAS</b> . . . . .	8
<b>Appendices</b> . . . . .	9
<b>Apêndice A Artigo publicado no SBSeg</b> . . . . .	10
<b>Apêndice B Artigo corrigido</b> . . . . .	11



## 1 INTRODUÇÃO

Este trabalho apresenta uma ferramenta desenvolvida para testes de intrusão em aplicações Web, o FuzzingTool. Inicialmente essa ferramenta foi projetada apenas para identificar vulnerabilidades de SQL injection, como parte de uma palestra ministrada na VI Semana de Tecnologia da Informação da Universidade Federal de Lavras em novembro de 2020 (BORGES, 2020).

No mesmo mês foi lançada a primeira *release* de correção de bugs, por meio de *feedback* de alunos da UFLA que testaram o FuzzingTool. Já em dezembro do mesmo ano, a funcionalidade de paralelismo estava em fase de implementação, ainda com problemas relacionados ao tratamento de exceções das requisições e condição de corrida entre as tarefas. Tais questões vieram a ser resolvidas em março de 2021, abrindo então espaço para a implementação de novas funcionalidades.

Em meados de junho de 2021, o professor Joaquim Uchôa sugeriu a elaboração de um artigo sobre a ferramenta para publicá-lo no Simpósio Brasileiro de Segurança da Informação que ocorreria ao fim do ano. O artigo foi submetido em julho, aceito em agosto e então publicado em outubro de 2021 no XXI SBSeg (BORGES; UCHÔA, 2021). A versão original deste trabalho se encontra no Apêndice A e a versão corrigida se encontra no Apêndice B. Ele contempla os seguintes pontos:

- motivação por trás da ferramenta;
- conceitos básicos como o método de ataque e protocolo de transferência utilizados;
- ferramentas relacionadas;
- descrição da ferramenta e seu modo de funcionamento;
- resultados obtidos;
- conclusão geral com sugestões para trabalhos futuros.

Para uma melhor análise da ferramenta foram realizados três testes, dois deles realizados em um ambiente providenciado propriamente para testes de intrusão. Com os resultados destes testes foi possível identificar vulnerabilidades de SQL Injection, arquivos e diretórios que normalmente não são acessados pelos usuários, e *sites* auxiliares que geralmente são alvos propícios a ataques.

Antes da publicação no SBSeg, a ferramenta passou para a *release* v3.12.0<sup>1</sup> e após a publicação, em janeiro deste ano, passou para a *release* v3.13.0<sup>2</sup>. Para a próxima versão, está prevista a entrada de funcionalidades como: recursão de tarefas (o que inclui recursão de arquivos e diretórios e recursão pelo enfileiramento de *payloads* dos *scanners*), nova configuração do Matcher para validar respostas por expressões regulares e a introdução de uma classe Filter para excluir respostas por códigos de status HTTP ou por expressões regulares.

---

<sup>1</sup> Release v3.12.0: <https://github.com/NESCAU-UFLA/FuzzingTool/releases/tag/v3.12.0>

<sup>2</sup> Release v3.13.0: <https://github.com/NESCAU-UFLA/FuzzingTool/releases/tag/v3.13.0>

## REFERÊNCIAS

BORGES, V. **SQLinjection - Dos pontos de vista do atacante e do defensor**. 2020. Disponível em: <<https://youtu.be/b0-La06i4FM>>.

BORGES, V.; UCHÔA, J. Fuzzingtool: Ferramenta para testes de intrusão em aplicações web. In: **Anais do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. Porto Alegre, RS, Brasil: SBC, 2021. p. 391–396. ISSN 0000-0000. Disponível em: <<https://sol.sbc.org.br/index.php/sbseg/article/view/17331>>.

# **Appendices**

**A ARTIGO PUBLICADO NO SBSEG**

# FuzzingTool: Ferramenta para Testes de Intrusão em Aplicações Web

Vitor O. C. N. Borges<sup>1</sup>, Joaquim Q. Uchôa<sup>1</sup>

<sup>1</sup>Departamento de Computação Aplicada (DAC)  
Instituto de Ciências Exatas e Tecnológicas (ICET)  
Universidade Federal de Lavras (UFLA)  
Caixa Postal 3037, CEP 37200-900 – Lavras/MG – Brazil

**Abstract.** *Every day, Web applications are being used to complex activities, in order to meet market demands. With the growing use and advance of its technologies, it requires to be more concerned with information security. For that, this article presents a tool developed for intrusion testing in Web applications, the FuzzingTool. The tool uses the fuzzing technique to find flaws in these applications, with very promising results in tests, e.g., identifying several potentially unsafe auxiliary sites from a main domain.*

**Resumo.** *A cada dia as aplicações Web estão sendo usadas para atividades complexas, a fim de atender as demandas de mercado. Com o crescente uso e avanço de suas tecnologias, faz-se necessário uma maior preocupação quanto à segurança da informação. Para tal, este artigo apresenta uma ferramenta desenvolvida para testes de intrusão em aplicações Web, o FuzzingTool. A ferramenta faz uso da técnica de fuzzing para localizar falhas nessas aplicações, com resultados bastante promissores em testes, por exemplo, que permitiram identificar vários sites auxiliares, potencialmente inseguros, a partir de um domínio principal.*

## 1. Introdução

Com o desenvolvimento contínuo da tecnologia Web, os navegadores, como o principal transporte de aplicativos e disseminação de conteúdo, fornecem aos usuários muitas conveniências: mecanismos de pesquisa, sites de redes sociais, compras online, transações online, etc. [Jingyu et al. 2021]. Até o início do século XXI, a maior parte de conteúdo era disponibilizado na forma de *landing pages* (i.e., páginas para apresentação, podendo conter formulários de contato). Atualmente, por outro lado, as páginas Web atendem parcialmente, senão completamente, toda a lógica de negócio de uma empresa, fato acelerado com a recente pandemia, em que diversas empresas optaram por mover seus negócios para aplicações Web.

Em virtude disso, manter um sistema Web seguro é imprescindível, seja por manter os serviços disponíveis e íntegros, informações sensíveis da empresa, ou pela proteção dos dados pessoais de clientes e funcionários. É importante ressaltar que as aplicações Web estão cada vez mais complexas, dinâmicas e com múltiplas funcionalidades. Diversos *frameworks* e tecnologias surgiram para tentar suprimir a necessidade de mercado; e, ao mesmo tempo, as vulnerabilidades acompanharam esse crescimento [Jackson 2021, Ferreira 2021].

Vulnerabilidades nas aplicações precisam ser identificadas e corrigidas para reduzir os riscos aos ativos da empresa, sendo que uma maneira bastante eficiente para essa tarefa é o uso de testes de intrusão. Em linhas gerais, um teste de intrusão visa identificar

e, em alguns casos, explorar uma vulnerabilidade; seja para ganhar acesso total ou parcial ao sistema, obter informações sensíveis ou mesmo negar o serviço. Nas aplicações Web não é diferente, com o adicional de que, por conta de suas características, há diversas formas de se realizar um teste de intrusão.

Neste artigo será apresentada uma ferramenta voltada a testes de intrusão em aplicações Web, o FuzzingTool<sup>1</sup>, disponibilizado por meio da licença MIT<sup>2</sup>. Antes, faremos breve introdução ao *fuzzing*, a seus conceitos básicos, e como essa técnica pode ser usada para identificar ou explorar vulnerabilidades nessas aplicações. Entre outras possibilidades de resultados da aplicação, tem-se, por exemplo, a identificação de sites auxiliares, potencialmente inseguros, a partir de um site principal.

## 2. Conceitos Básicos

FuzzingTool é uma ferramenta voltada a testes de intrusão em aplicações Web, utilizando *fuzzing* como modo de operação. *Fuzzing*, por sua vez é uma das técnicas utilizadas por profissionais da área de segurança da informação, inicialmente projetada para identificar *crashes* (*i.e.*, interrupções inesperadas do serviço em execução) em sistemas UNIX [Miller et al. 1990]. Seu objetivo consiste em fazer requisições ao alvo por força-bruta, em formato *black-box* (*i.e.*, a construção dos dados de entrada é feita sem o conhecimento prévio do alvo), enviando parâmetros totalmente ou parcialmente malformados para a aplicação [Forrester and Miller 2000]. Ao realizar tais requisições, a aplicação pode responder de diversas maneiras, incluindo exceções ou mesmo com informações sensíveis sobre o alvo, ou mesmo interrompendo o seu serviço.

*A simplicidade desta técnica significa que ela tem um baixo custo e é fácil de se aplicar [...]. Nosso artigo de 1990 observou que a única falha de maior categoria no dia era o acesso fora dos limites de um buffer e sugeri que o teste de difusão poderia ser usado [...] para ajudar a encontrar falhas de segurança”. Por mais simples que seja, ainda é o tipo de erro que continua a causar interrupções massivas na Internet, conforme evidenciado pela vulnerabilidade Heartbleed no OpenSSL em 2014 e a vulnerabilidade mais recente no clássico comando sudo em 2018 [Miller et al. 2020].*

Do ponto de vista da Web, essa técnica pode ser aplicada a uma infinidade de cenários. Requisições Web carregam consigo, obrigatoriamente, um cabeçalho e, em alguns casos, também carregam um corpo; as respostas a tais requisições funcionam da mesma forma, com a diferença de que quase sempre trazem um corpo. Enquanto o corpo da requisição serve para enviar dados ao servidor, o corpo da resposta contém a própria página buscada. Entender como essas requisições são feitas, e como são respondidas, se torna o ponto chave para se tirar maior proveito da técnica de *fuzzing*.

Os vetores de ataque mais comuns utilizados por essa técnica são os campos de formulários e parâmetros de requisições GET. No geral, procuram-se vulnerabilidades como SQL Injection, XSS (*i.e.*, *Cross-Site Scripting*) e *path traversal*, listadas entre as maiores vulnerabilidades Web [The OWASP Foundation 2020]. Contudo a abordagem não se limita a apenas tais vetores, podendo também ser aplicada à enumeração dos métodos aceitos pela requisição, à enumeração dos caminhos da requisição (arquivos e diretórios), e à injeção de código nos cabeçalhos (*e.g.*, dentro dos *cookies*).

---

<sup>1</sup>FuzzingTool: <https://github.com/NESCAU-UFLA/FuzzingTool>.

<sup>2</sup>Licença utilizada: <https://github.com/NESCAU-UFLA/FuzzingTool/blob/master/LICENSE>.

Pela sua natureza de ser um método exaustivo, muita informação é gerada; separar apenas as respostas de interesse é fundamental para se evitar falsos positivos. Entram no escopo da análise, dependendo do que se está testando, as seguintes informações: o código de status, latência, tamanho do corpo, conteúdo do corpo, URL de destino, redirecionamentos e cabeçalho. Por exemplo, ao procurar por vulnerabilidades de *Cross-Site Scripting*, deve-se investigar no corpo da resposta se o *payload* refletiu integralmente, ou seja, sem que tenha ocorrido nenhuma remoção ou escape de caracteres especiais.

Dentre as ferramentas gratuitas de *fuzzing* para aplicações Web, existem várias aplicações similares, entretanto, são poucas aquelas que tenham sido favoritadas por pelo menos 1000 usuários no Github, com recursos para modificar parâmetros de ataque e adicionar opções de requisição e afins. Das que atendem esses critérios, foram identificadas apenas as seguintes aplicações: Wfuzz<sup>3</sup> e Ffuf<sup>4</sup> que, assim como a ferramenta apresentada neste trabalho, prestam funcionalidades gerais de *fuzzing*; dirsearch<sup>5</sup> para enumeração de arquivos e diretórios; Gobuster<sup>6</sup> para enumeração de *hosts* virtuais, DNS e arquivos e diretórios. O diferencial do FuzzingTool para as ferramentas mencionadas é sua capacidade de construir o resultado de acordo com o tipo de ataque feito, além da apresentação desses resultados no terminal. Por exemplo, ao se realizar uma enumeração de subdomínios, o endereço IP do alvo é adicionado ao resultado. Desse modo podemos ter resultados com pontos específicos de interesse, sem perder dados importantes sobre os testes feitos.

### 3. A Ferramenta

FuzzingTool foi desenvolvido em Python 3.6, para uso em ambientes Linux, por meio de interface de linha de comando. Para o desenvolvimento, foi utilizada orientação a objetos e alguns padrões de projeto, como *decorator*, *factory* e *strategy* [Refactoring Guru 2014]. Inicialmente foi projetada apenas para testes de Cross-Site Scripting e SQL Injection em aplicações Web locais, de maneira sequencial. Atualmente, ela permite o *fuzzing* de dados (corpo e parâmetros da requisição), enumeração dos métodos da requisição, enumeração dos arquivos e diretórios existentes na aplicação, e enumeração dos subdomínios do alvo que contenham um servidor Web ativo, além de realizar requisições em paralelo. Na Figura 1, é apresentado seu modo de funcionamento.

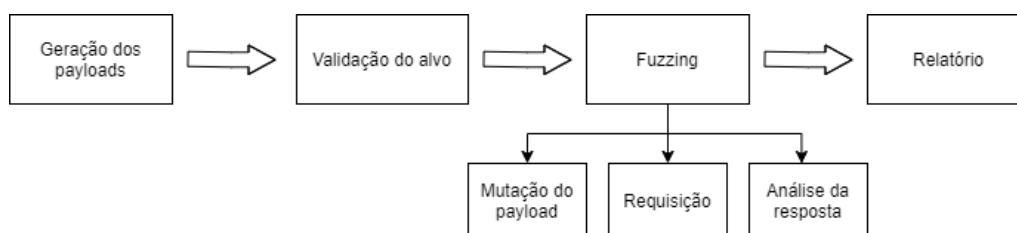


Figura 1. Modo de funcionamento

Quando da chamada da aplicação, inicialmente é verificado se todos os argumentos foram instanciados corretamente. Após, ocorre o processo de geração dos *payloads*, encarregado por gerá-los e também definir suas regras de mutação. Antes de iniciar o *fuzzing*, o alvo em questão é validado ao realizar um teste de conexão (e redirecionamentos

<sup>3</sup>Wfuzz: <https://wfuzz.readthedocs.io/en/latest/>.

<sup>4</sup>Ffuf: <https://github.com/ffuf/ffuf>.

<sup>5</sup>dirsearch: <https://github.com/maurosoria/dirsearch>.

<sup>6</sup>Gobuster: <https://github.com/OJ/gobuster>.



para o caso de *fuzzing* de dados). Já na fase de *fuzzing* é onde se encontra o núcleo da ferramenta, dividindo-se em três partes: a mutação do *payload*, a requisição e a análise da resposta. Por fim há a etapa de escrita dos dados obtidos num relatório.

Na etapa de geração dos *payloads*, é criado um objeto Dictionary, globalmente ou para cada alvo, contendo uma ou mais *wordlists*. Tais *wordlists* podem ser geradas a partir de um arquivo, uma lista de palavras ou mesmo geradas por *plugins*. Esse objeto Dictionary utiliza os métodos estáticos de uma outra classe denominada Payloader, responsável por realizar a mutação dos *payloads* durante a fase de *fuzzing*. Esse Payloader detém as configurações de prefixos, sufixos, caixa alta e baixa, e dos *encoders*. O conteúdo das *wordlists* é inserido numa fila, que será consumida pelo processamento mais adiante.

Na fase de *fuzzing*, ocorre o processamento do programa. O Fuzzer - objeto responsável pelo *fuzzing* - consome a fila de *payloads* do Dictionary (com as regras de mutação), enviando essa carga à requisição e, ao receber uma resposta, ela é tratada num objeto Resultado, para então ocorrer a sua análise. Esse resultado por sua vez contém informações básicas, presentes em todos os tipos de testes: o *payload* utilizado na requisição, a URL do alvo, o tamanho do corpo da resposta (em *bytes*), a quantidade de linhas e palavras no corpo da resposta e a latência.

A análise é feita em duas partes aninhadas (*i.e.*, para que a segunda análise seja feita, ela depende do resultado da primeira), com a intenção de diminuir falsos positivos. À primeira instância, é verificado pelo objeto Matcher se um conjunto de regras estabelecidas pelo usuário se aplicam ao resultado, tais como: códigos de *status* HTTP, tamanho do corpo da resposta (em *bytes*) e a latência. Em sequência o objeto Scanner, escolhido automaticamente pelo programa ou por um *plugin*, pode ou não efetuar uma avaliação em cima do resultado.

Cada Scanner, além de fazer essa avaliação adicional, faz ao menos uma adição de informação ao objeto Resultado, e define o *callback* para a escrita desse resultado à interface de saída. O PathScanner, um dos *scanners* padrões da ferramenta, adiciona a informação da URL de redirecionamento nos casos em que o código de status 301 e 302 são detectados. No *callback* de escrita, ele faz a coloração dos códigos de *status* para facilitar a identificação de páginas com acesso negado, acessíveis ou com redirecionamentos.

Existem alguns recursos para desvio de *firewall* e WAF (*Web Application Firewall*) [Yari et al. 2019], tais como: alterar os cabeçalhos da requisição por completo (ao invés de passar uma URL como alvo para a ferramenta, é passado um arquivo que contenha as suas informações de requisição), fazer a mutação do *payload*, definir um *delay* entre cada requisição, e usar *proxies*.

Há a opção de colocar códigos de *status* HTTP em uma lista negra. Com isso, é possível que o programa execute uma ação específica ao detectá-los, como: pular o alvo atual ou pausar a execução por um período de tempo. Tal funcionalidade se torna interessante quando, por exemplo, queremos evitar a sobrecarga do alvo ao colocar na lista o código 500 (*Internal Server Error*); ou mesmo quando queremos pausar a ferramenta caso o alvo venha a rejeitar requisições, colocando na lista o código 429 (*Too Many Requests*).

O aplicativo dispõe de mais de 30 argumentos a serem usados nos testes, sendo os principais o *-u* para especificar o alvo e o *-w* para especificar *wordlists*. Outras opções que merecem ser listadas são o *-X* para definir o método da requisição e o *-t* para definir a quantidade de *threads* que serão usadas. Para uma completa verificação dos argumentos disponíveis, pode-se usar a opção *-h* ou *-help*.

#### 4. Testes e Resultados

Para uma melhor análise da ferramenta, foram realizados três testes, dois deles realizados em um ambiente providenciado propriamente para testes de intrusão<sup>7</sup>. O primeiro teste foi realizado para identificar vulnerabilidades de SQL Injection [Sivasangari et al. 2021] em uma página de *login*. Nesse teste, foi realizado *fuzzing* de dados e, quando um comparador de dados não é especificado ao Matcher, o próprio programa tenta calcular e sugerir ao usuário as informações para este comparador. Esse cálculo é feito a partir de uma requisição à página alvo, coletando informações como o tamanho do corpo da resposta em *bytes*, e a latência dela. A *wordlist* utilizada foi construída objetivamente para testes gerais de SQL Injection, contendo 1358 *payloads*.

Como pode ser verificado na Tabela 1, foram encontrados 22 resultados para o ataque Boolean-Based que atenderam às regras do Matcher. Em contrapartida, os testes para o Time-Based tiveram resultados discrepantes ao variarmos os valores da configuração do Matcher. Isso se dá pois a latência de uma requisição Web pode variar de acordo com o estado da conexão tanto do cliente quanto do servidor. Por esse motivo, ajustar a configuração para atender a uma latência maior trará, dependendo da *wordlist*, alguns falsos negativos, porém, com uma maior segurança.

**Tabela 1. Resultados para o teste de SQL Injection**

Tipo de ataque <i>Blind SQLi</i>	Configuração do Matcher	Resultados encontrados	Falsos-positivos
<i>Boolean-Based</i>	Tamanho do corpo da resposta > 5823 bytes	22	0%
<i>Time-Based</i>	Latência > 2 seg	73	97,26%
<i>Time-Based</i>	Latência > 10 seg	1	0%

O segundo teste foi feito para enumerar arquivos e diretórios da aplicação. A *wordlist* utilizada continha 4780 *payloads*, com arquivos que, normalmente, não deveriam estar expostos, como: arquivos de configuração do Apache, do WordPress e de projetos do Github. Pela natureza do teste, deseja-se encontrar os arquivos e diretórios existentes que são normalmente acessíveis, os que levam a redirecionamentos, e os que têm acesso restrito. Para isso foram incluídos os códigos de *status* 200, 301, 302, 401 e 403 nas regras do Matcher. Foram encontradas 19 páginas com *status* 200, duas páginas com *status* 301 e uma com *status* 403.

O terceiro teste consistiu em enumerar *hosts* (subdomínios) que teriam o serviço HTTP ativo, de duas instituições federais de ensino superior. Em cada alvo foi utilizado o *plugin* DnsZone para a construção da sua respectiva *wordlist*, obtendo os *payloads* para o teste a partir da exploração da vulnerabilidade de transferência de zona irrestrita [CVE 1999]. Na primeira instituição, foi gerada uma *wordlist* com 993 *payloads*, encontrando 151 *hosts* com o serviço ativo. Para a segunda instituição, foi gerada uma *wordlist* com 453 *payloads*, encontrando 63 *hosts* com o serviço ativo.

<sup>7</sup>Acunetix Acuart: <http://testphp.vulnweb.com/>

Com os resultados obtidos, percebe-se o potencial da ferramenta. No caso das instituições de ensino, por exemplo, geralmente é considerado que o *site* principal é relativamente vigiado e seguro. Entretanto, os *sites* auxiliares geralmente não recebem a devida atenção, sendo alvo fácil de ataque, possibilitando a escalada de acesso a recursos computacionais das instituições.

## 5. Conclusão

Dado o crescente desenvolvimento das tecnologias em aplicações Web, torna-se cada vez mais importante se preocupar quanto à sua segurança. Neste artigo foi apresentado como a técnica de *fuzzing* é aplicada para testes de intrusão nessas aplicações, com o foco em uma ferramenta para lidar com esses testes, listando seus modos de operação e um exemplo de uso.

Para trabalhos futuros, alguns já em andamento, pretende-se: adicionar uma interface Web à ferramenta, utilizando a biblioteca Flask do Python; adicionar mutadores dos *payloads* voltados à linguagens específicas (*e.g.*, SQL, JavaScript); adicionar novos *encoders* (*e.g.*, um para tratar códigos hexadecimais em HTML, denominado HtmlHexadecimal); adicionar um mecanismo de detecção e prevenção a *honeypots*; inserir a ferramenta no repositório oficial do Kali Linux<sup>8</sup>.

## Referências

- CVE (1999). CVE-1999-0532. Disponível em: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0532>. Acesso em: 06 de Julho, 2021.
- Ferreira, N. (2021). Joomla JCK Editor 6.4.4 - 'parent' SQL Injection (2). Disponível em: <https://www.exploit-db.com/exploits/49627>. Acesso em: 06 de Julho, 2021.
- Forrester, J. E. and Miller, B. P. (2000). An empirical study of the robustness of Windows NT applications using random testing. *4th USENIX Windows Systems Symposium*.
- Jackson, T. (2021). Wordpress Plugin YOP Polls 6.2.7 - Stored Cross Site Scripting (XSS). Disponível em: <https://www.exploit-db.com/exploits/50066>. Acesso em: 06 de Julho, 2021.
- Jingyu, Z., Hongchao, H., Shumin, H., and Huanruo, L. (2021). A XSS attack detection method based on subsequence matching algorithm. In *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*, pages 83–86.
- Miller, B., Zhang, M., and Heymann, E. (2020). The relevance of classic fuzz testing: Have we solved this one? *IEEE Transactions on Software Engineering*, pages 1–1.
- Miller, B. P., Fredriksen, L., and So, B. (1990). An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, (33).
- Refactoring Guru (2014). Padrões de projeto em python. Disponível em: <https://refactoring.guru/pt-br/design-patterns/python>. Acesso em: 06 de Julho, 2021.
- Sivasangari, A., Jyotsna, J., and Pravalika, K. (2021). SQL injection attack detection using machine learning algorithm. In *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 1166–1169.
- The OWASP Foundation (2020). Top 10 web application security risks. Disponível em: <https://owasp.org/www-project-top-ten/>. Acesso em: 06 de Julho, 2021.
- Yari, I. A., Abdullahi, B., and Adeshina, S. A. (2019). Towards a framework of configuring and evaluating modsecurity waf on tomcat and apache web servers. In *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1–7.

---

<sup>8</sup>Kali Linux: <https://www.kali.org/>

**B ARTIGO CORRIGIDO**

# FuzzingTool: Ferramenta para Testes de Intrusão em Aplicações Web

Vitor O. C. N. Borges<sup>1</sup>, Joaquim Q. Uchôa<sup>2</sup>

<sup>1</sup>Departamento de Computação Aplicada (DAC)  
Instituto de Ciências Exatas e Tecnológicas (ICET)  
Universidade Federal de Lavras (UFLA)  
Caixa Postal 3037, CEP 37200-900 – Lavras/MG – Brazil

**Abstract.** *Every day, Web applications are being used to complex activities, in order to meet market demands. With the growing use and advance of its technologies, it requires to be more concerned with information security. For that, this article presents a tool developed for intrusion testing in Web applications, the FuzzingTool. The tool uses the fuzzing technique to find flaws in these applications, with very promising results in tests, e.g., identifying several potentially unsafe auxiliary sites from a main domain.*

**Resumo.** *A cada dia as aplicações Web estão sendo usadas para atividades complexas, a fim de atender as demandas de mercado. Com o crescente uso e avanço de suas tecnologias, faz-se necessário uma maior preocupação quanto à segurança da informação. Para tal, este artigo apresenta uma ferramenta desenvolvida para testes de intrusão em aplicações Web, o FuzzingTool. A ferramenta faz uso da técnica de fuzzing para localizar falhas nessas aplicações, com resultados bastante promissores em testes, por exemplo, que permitiram identificar vários sites auxiliares, potencialmente inseguros, a partir de um domínio principal.*

## 1. Introdução

Até o início do século XXI, a maior parte de conteúdo nas aplicações Web era disponibilizado na forma de *landing pages* (i.e., páginas para apresentação, podendo conter formulários de contato). Atualmente, por outro lado, essas páginas atendem parcialmente, senão completamente, toda a lógica de negócio de uma empresa. Com o desenvolvimento contínuo da tecnologia Web, os navegadores, como o principal meio de disseminação de conteúdo, fornecem aos usuários muitas conveniências: mecanismos de pesquisa, sites de redes sociais, compras online, transações online, etc. [Jingyu et al. 2021]. Este fato foi acelerado com a recente pandemia, em que diversas empresas optaram por mover seus negócios para aplicações Web.

Em virtude disso, manter um sistema Web seguro é imprescindível, seja por manter os serviços disponíveis e íntegros, informações sensíveis da empresa, ou pela proteção dos dados pessoais de clientes e funcionários. É importante ressaltar que as aplicações Web estão cada vez mais complexas, dinâmicas e com múltiplas funcionalidades. Diversos *frameworks* e tecnologias surgiram para tentar suprimir a necessidade de mercado; e, ao mesmo tempo, as vulnerabilidades acompanharam esse crescimento [Jackson 2021, Ferreira 2021].

Por conta do crescente uso dessas aplicações, suas vulnerabilidades precisam ser identificadas e corrigidas para reduzir os riscos aos ativos da empresa, sendo que uma

maneira bastante eficiente para essa tarefa é o uso de testes de intrusão. Em linhas gerais, um teste de intrusão visa avaliar a qualidade dos mecanismos de segurança por meio da identificação e, em alguns casos, exploração de uma vulnerabilidade; seja para ganhar acesso total ou parcial ao sistema, obter informações sensíveis ou mesmo negar o serviço. Nas aplicações Web não é diferente, com o adicional de que, por conta de suas características, há diversas formas de se realizar um teste de intrusão.

Neste artigo será apresentada uma ferramenta voltada a testes de intrusão em aplicações Web, o FuzzingTool<sup>1</sup>, disponibilizado por meio da licença MIT<sup>2</sup>. Antes, faremos breve introdução ao *fuzzing*, a seus conceitos básicos, e como essa técnica pode ser usada para identificar ou explorar vulnerabilidades nessas aplicações. Entre outras possibilidades de resultados da aplicação, tem-se, por exemplo, a identificação de sites auxiliares, potencialmente inseguros, a partir de um domínio principal.

## 2. Conceitos Básicos

FuzzingTool é uma ferramenta voltada a testes de intrusão em aplicações Web, utilizando *fuzzing* como modo de operação. O *Fuzzing* é uma técnica utilizada por profissionais da área de segurança da informação, inicialmente projetada para identificar *crashes* (*i.e.*, interrupções inesperadas do serviço em execução) em sistemas UNIX [Miller et al. 1990]. O objetivo de utilização desta técnica consiste em fazer requisições ao alvo por força-bruta, em formato *black-box* (*i.e.*, a construção dos dados de entrada é feita sem o conhecimento prévio do alvo), enviando parâmetros totalmente ou parcialmente malformados para a aplicação [Forrester and Miller 2000]. Ao realizar tais requisições, a aplicação pode responder de diversas maneiras, incluindo exceções ou mesmo com informações sensíveis sobre o alvo, ou mesmo interrompendo o seu serviço [Miller et al. 2020].

Do ponto de vista da Web, essa técnica pode ser aplicada a uma infinidade de cenários. Requisições HTTP carregam consigo, obrigatoriamente, um cabeçalho e, em alguns casos, também carregam um corpo; as respostas a tais requisições funcionam da mesma forma, com a diferença de que quase sempre trazem um corpo. Enquanto o corpo da requisição serve para enviar dados ao servidor, o corpo da resposta contém a própria página buscada. Entender como essas requisições são feitas, e como são respondidas, se torna o ponto chave para se tirar maior proveito da técnica de *fuzzing*.

Os vetores de ataque mais comuns utilizados por essa técnica são os campos de formulários e parâmetros de requisições GET. No geral, procuram-se vulnerabilidades como SQL Injection, XSS (*i.e.*, *Cross-Site Scripting*) e *path traversal*, listadas entre as maiores vulnerabilidades Web [The OWASP Foundation 2020]. Contudo a abordagem não se limita a apenas tais vetores, podendo também ser aplicada à enumeração dos métodos aceitos pela requisição, à enumeração dos caminhos da requisição (arquivos e diretórios), e à injeção de código nos cabeçalhos (*e.g.*, dentro dos *cookies*).

Pela natureza do *fuzzing* ser um método exaustivo, muita informação é gerada; separar apenas as respostas de interesse é fundamental para se evitar falsos positivos. Entram no escopo da análise, dependendo do que se está testando, as seguintes informações: o código de status, latência, tamanho do corpo, conteúdo do corpo, URL de destino, redirecionamentos e cabeçalho. Por exemplo, ao procurar por vulnerabilidades de *Cross-Site Scripting*, um indicador de sua presença pode ser encontrado no corpo da resposta, caso o

---

<sup>1</sup>FuzzingTool: <https://github.com/NESCAU-UFLA/FuzzingTool>.

<sup>2</sup>Licença utilizada: <https://github.com/NESCAU-UFLA/FuzzingTool/blob/master/LICENSE>.

*payload* tenha refletido integralmente, ou seja, sem que tenha ocorrido nenhuma remoção ou escape de caracteres especiais.

Dentre as ferramentas gratuitas de *fuzzing* para aplicações Web, existem várias aplicações similares, entretanto, são poucas aquelas que tenham sido favoritadas por pelo menos 1000 usuários no Github, com recursos para modificar parâmetros de ataque e adicionar opções de requisição e afins. Das que atendem esses critérios, foram identificadas apenas as seguintes aplicações: Wfuzz<sup>3</sup> e Ffuf<sup>4</sup> que, assim como a ferramenta apresentada neste trabalho, prestam funcionalidades gerais de *fuzzing*; dirsearch<sup>5</sup> para enumeração de arquivos e diretórios; Gobuster<sup>6</sup> para enumeração de *hosts* virtuais, DNS e arquivos e diretórios. Uma comparação de funcionalidades entre as ferramentas pode ser vista na Tabela 1.

**Tabela 1. Comparação de funcionalidades entre as ferramentas**

	FuzzingTool	Wfuzz	Ffuf	Dirsearch	Gobuster
Permite salvar o relatório em vários formatos	X		X	X	
Gera o relatório de forma detalhada	X		X	X	
Permite modificar o cabeçalho HTTP	X	X	X	X	X
Permite a mutação do payload	X	X	X	X	X
Suporte a plugins	X	X			

### 3. A Ferramenta

FuzzingTool foi desenvolvido em Python 3.6, para uso em ambientes Linux, por meio de interface de linha de comando. Para o desenvolvimento, foi utilizada orientação a objetos e alguns padrões de projeto, como *decorator*, *factory* e *strategy* [Refactoring Guru 2014]. Inicialmente foi projetada apenas para testes de Cross-Site Scripting e SQL Injection em aplicações Web locais, de maneira sequencial. Atualmente, ela permite o *fuzzing* de dados (corpo e parâmetros da requisição), enumeração dos métodos da requisição, enumeração dos arquivos e diretórios existentes na aplicação, e enumeração dos subdomínios do alvo que contenham um servidor Web ativo, além de realizar requisições em paralelo. Na Figura 1, é apresentado seu modo de funcionamento.

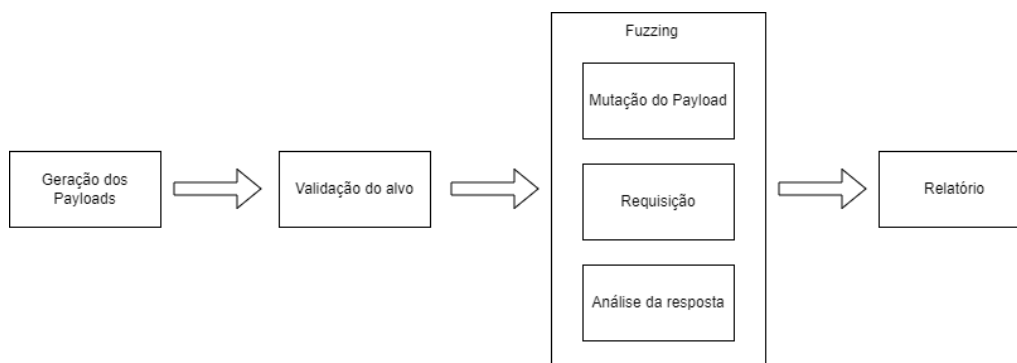
Quando da chamada da aplicação, inicialmente é verificado se todos os argumentos foram instanciados corretamente. Após, ocorre o processo de geração dos *payloads*, encarregado por gerá-los e também definir suas regras de mutação. Antes de iniciar o *fuzzing*, o alvo em questão é validado ao realizar um teste de conexão (e redirecionamentos para o caso de *fuzzing* de dados). Já na fase de *fuzzing* é onde se encontra o núcleo da

<sup>3</sup>Wfuzz: <https://wfuzz.readthedocs.io/en/latest/>.

<sup>4</sup>Ffuf: <https://github.com/ffuf/ffuf>.

<sup>5</sup>dirsearch: <https://github.com/maurosoria/dirsearch>.

<sup>6</sup>Gobuster: <https://github.com/OJ/gobuster>.



**Figura 1. Modo de funcionamento**

ferramenta, dividindo-se em três partes: a mutação do *payload*, a requisição e a análise da resposta. Por fim há a etapa de escrita dos dados obtidos num relatório.

### 3.1. Geração dos payloads

Na etapa de geração dos *payloads* é criado um objeto Dictionary composto por uma ou mais *wordlists* que fornecem os payloads, geradas a partir de um arquivo, uma lista de palavras ou mesmo por *plugins*. Os *payloads* gerados são enfileirados no Dictionary para serem consumidos na etapa seguinte. O Dictionary utiliza os métodos estáticos de uma outra classe denominada *Payloader*, responsável por realizar a mutação dos *payloads* durante a fase de *fuzzing*. Esse *Payloader* detém as configurações de prefixos, sufixos, caixa alta e baixa, e dos *encoders*.

### 3.2. Fuzzing

Na fase de *fuzzing* ocorre o processamento do programa. O *Fuzzer* - objeto responsável pelo *fuzzing* - consome a fila de *payloads* do Dictionary (com as regras de mutação), enviando essa carga à requisição e, ao receber uma resposta, ela é salva num objeto de resultado para posterior análise. O resultado por sua vez contém informações básicas, presentes em todos os tipos de testes: o *payload* utilizado na requisição, a URL do alvo, o tamanho do corpo da resposta (em *bytes*), a quantidade de linhas e palavras no corpo da resposta e a latência.

A análise é feita em duas partes aninhadas (*i.e.*, para que a segunda análise seja feita, ela depende do resultado da primeira), com a intenção de diminuir falsos positivos. A primeira parte é verificada pelo objeto *Matcher* se um conjunto de regras estabelecidas pelo usuário se aplicam ao resultado, tais como: códigos de *status* HTTP, tamanho do corpo da resposta (em *bytes*) e a latência. A segunda parte consiste na execução do objeto *Scanner*, escolhido automaticamente pelo programa ou por um *plugin*, pode ou não efetuar uma avaliação em cima do resultado.

Cada *Scanner*, além de fazer essa avaliação adicional, faz ao menos uma adição de informação ao objeto *Resultado*, e define o *callback* para a escrita desse resultado à interface de saída. O *PathScanner* é um dos *scanners* padrões da ferramenta, adiciona a informação da URL de redirecionamento nos casos em que o código de *status* 301 e 302 são detectados. No *callback* de escrita, ele faz a coloração dos códigos de *status* para facilitar a identificação de páginas com acesso negado, acessíveis ou com redirecionamentos.

### 3.3. Recursos Adicionais

A ferramenta disponibiliza ainda alguns recursos para desvio de *firewall* e WAF (*Web Application Firewall*) [Yari et al. 2019], tais como: alterar os cabeçalhos da requisição



por completo (ao invés de passar uma URL como alvo para a ferramenta, é passado um arquivo que contenha as suas informações de requisição), fazer a mutação do *payload*, definir um *delay* entre cada requisição, e usar *proxies*.

Há a opção de colocar códigos de *status* HTTP em uma lista negra. Com isso, é possível que o programa execute uma ação específica ao detectá-los, como: pular o alvo atual ou pausar a execução por um período de tempo. Tal funcionalidade se torna interessante quando, por exemplo, queremos evitar a sobrecarga do alvo ao colocar na lista o código 500 (*Internal Server Error*); ou mesmo quando queremos pausar a ferramenta caso o alvo venha a rejeitar requisições, colocando na lista o código 429 (*Too Many Requests*).

O aplicativo dispõe de mais de 30 argumentos a serem usados nos testes, sendo os principais o *-u* para especificar o alvo e o *-w* para especificar *wordlists*. Outras opções que merecem ser listadas são o *-X* para definir o método da requisição e o *-t* para definir a quantidade de *threads* que serão usadas. Para uma completa verificação dos argumentos disponíveis, pode-se usar a opção *-h* ou *-help*.

#### 4. Testes e Resultados

Para uma melhor análise da ferramenta foram realizados três testes, dois deles realizados em um ambiente providenciado propriamente para testes de intrusão<sup>7</sup>. O primeiro teste foi realizado para identificar vulnerabilidades de SQL Injection [Sivasangari et al. 2021] em uma página de *login*. Nesse teste, foi realizado *fuzzing* de dados e, quando um comparador de dados não é especificado ao *Matcher*, o próprio programa tenta calcular e sugerir ao usuário as informações para este comparador. Esse cálculo é feito a partir de uma requisição à página alvo, coletando informações como o tamanho do corpo da resposta em *bytes*, e a latência dela. A *wordlist* utilizada foi construída objetivamente para testes gerais de SQL Injection, contendo 1358 *payloads*.

Como pode ser verificado na Tabela 2, foram encontrados 22 resultados para o ataque Boolean-Based que atenderam às regras do *Matcher*. Em contrapartida, os testes para o Time-Based tiveram resultados discrepantes ao variarmos os valores da configuração do *Matcher*. Isso se dá pois a latência de uma requisição Web pode variar de acordo com o estado da conexão tanto do cliente quanto do servidor. Por esse motivo, ajustar a configuração para atender a uma latência maior trará, dependendo da *wordlist*, alguns falsos negativos, porém, com uma maior segurança.

O segundo teste foi feito para enumerar arquivos e diretórios da aplicação, descrito como *Path Enumeration* na Tabela 2. A *wordlist* utilizada continha 4780 *payloads*, com arquivos que, normalmente, não deveriam estar expostos, como: arquivos de configuração do Apache, do WordPress e de projetos do Github. Pela natureza do teste, deseja-se encontrar os arquivos e diretórios existentes que são normalmente acessíveis, os que levam a redirecionamentos e os que têm acesso restrito. Para isso foram incluídos os códigos de *status* 200, 301, 302, 401 e 403 nas regras do *Matcher*. Foram encontradas 19 páginas com *status* 200, duas páginas com *status* 301 e uma com *status* 403.

O terceiro teste consistiu em enumerar *hosts* (subdomínios) que teriam o serviço HTTP ativo de duas instituições federais de ensino superior, descrito na Tabela 2 como *Subdomain Enumeration*. Em cada alvo foi utilizado o *plugin* *DnsZone* para a construção da sua respectiva *wordlist*, obtendo os *payloads* para o teste a partir da exploração da vulnerabilidade de transferência de zona irrestrita [CVE 1999]. Na primeira instituição,

---

<sup>7</sup>Acunetix Acuart: <http://testphp.vulnweb.com/>

foi gerada uma *wordlist* com 993 *payloads*, encontrando 151 *hosts* com o serviço ativo. Para a segunda instituição, foi gerada uma *wordlist* com 453 *payloads*, encontrando 63 *hosts* com o serviço ativo.

**Tabela 2. Resultados para os testes**

Tipo de ataque	Alvo	Configuração do Matcher	Resultados encontrados	Falsos-positivos
<i>Blind SQLi Boolean-Based</i>	Acunetix Acuart	Tamanho do corpo da resposta > 5823 bytes	22	0%
<i>Blind SQLi Time-Based</i>	Acunetix Acuart	Latência > 2 seg	73	97,26%
<i>Blind SQLi Time-Based</i>	Acunetix Acuart	Latência > 10 seg	1	0%
<i>Path Enumeration</i>	Acunetix Acuart	Código de status 200 a 399, 401 e 403	22	0%
<i>Subdomain Enumeration</i>	Instituição de Ensino 1	Código de status 200 a 399, 401 e 403	151	0%
<i>Subdomain Enumeration</i>	Instituição de Ensino 2	Código de status 200 a 399, 401 e 403	63	0%

## 5. Conclusão

Dado o crescente desenvolvimento das tecnologias em aplicações Web, torna-se cada vez mais importante se preocupar quanto aos aspectos de segurança computacional nesses sistemas. Com os resultados obtidos, percebe-se o potencial da ferramenta. No caso das instituições de ensino, por exemplo, geralmente é considerado que o *site* principal é relativamente vigiado e seguro. Entretanto, os *sites* auxiliares geralmente não recebem a devida atenção, sendo alvo propício de ataque, possibilitando a escalada de acesso a recursos computacionais das instituições. Neste artigo foi apresentado como a técnica de *fuzzing* é aplicada para testes de intrusão nessas aplicações, com o foco em uma ferramenta para lidar com esses testes, listando seus modos de operação e um exemplo de uso.

Para trabalhos futuros, alguns já em andamento, pretende-se: adicionar uma interface Web à ferramenta, utilizando a biblioteca Flask do Python; adicionar mutadores dos *payloads* voltados à linguagens específicas (*e.g.*, SQL, JavaScript); adicionar novos *encoders* (*e.g.*, um para tratar códigos hexadecimais em HTML, denominado HtmlHexadecimal); bem como inserir a ferramenta no repositório oficial do Kali Linux<sup>8</sup>.

<sup>8</sup>Kali Linux: <https://www.kali.org/>

## Referências

- CVE (1999). CVE-1999-0532. Disponível em: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=1999-0532>. Acesso em: 06 de Julho, 2021.
- Ferreira, N. (2021). Joomla JCK Editor 6.4.4 - 'parent' SQL Injection (2). Disponível em: <https://www.exploit-db.com/exploits/49627>. Acesso em: 06 de Julho, 2021.
- Forrester, J. E. and Miller, B. P. (2000). An empirical study of the robustness of Windows NT applications using random testing. *4th USENIX Windows Systems Symposium*.
- Jackson, T. (2021). Wordpress Plugin YOP Polls 6.2.7 - Stored Cross Site Scripting (XSS). Disponível em: <https://www.exploit-db.com/exploits/50066>. Acesso em: 06 de Julho, 2021.
- Jingyu, Z., Hongchao, H., Shumin, H., and Huanruo, L. (2021). A XSS attack detection method based on subsequence matching algorithm. In *2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID)*, pages 83–86.
- Miller, B., Zhang, M., and Heymann, E. (2020). The relevance of classic fuzz testing: Have we solved this one? *IEEE Transactions on Software Engineering*, pages 1–1.
- Miller, B. P., Fredriksen, L., and So, B. (1990). An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, (33).
- Refactoring Guru (2014). Padrões de projeto em python. Disponível em: <https://refactoring.guru/pt-br/design-patterns/python>. Acesso em: 06 de Julho, 2021.
- Sivasangari, A., Jyotsna, J., and Pravalika, K. (2021). SQL injection attack detection using machine learning algorithm. In *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 1166–1169.
- The OWASP Foundation (2020). Top 10 web application security risks. Disponível em: <https://owasp.org/www-project-top-ten/>. Acesso em: 06 de Julho, 2021.
- Yari, I. A., Abdullahi, B., and Adeshina, S. A. (2019). Towards a framework of configuring and evaluating modsecurity waf on tomcat and apache web servers. In *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*, pages 1–7.