Automação de Autenticação para Testes de Segurança em Aplicações Web no ZAP

Lucas S. C. Sacramento[1] **Ítalo Cunha**[1] **Gabriel Pains de Oliveira Cardoso**[1]
Artur Souza[2] **Antônio Franco**[1] **Leonardo B. Oliveira**[1]

[1] Departamento de Ciência da Computação — Universidade Federal de Minas Gerais

[2]Kider Security

{lucas.sacramento,cunha,gabriel.cardoso,franco,leob}@dcc.ufmg.br
arturluis@kidersecurity.com

Abstract. *Running exhaustive vulnerability tests on Web applications requires an authenticated user session. Authentication is usually configured manually by a security expert and is challenging due to the many different authentication methods used by modern Web apps. In this paper we build a framework that automatically builds the necessary configuration for automating authentication in Web app analysis tools. Our evaluation shows that the framework is both resource-efficient and effective at authenticating in many Web apps.*

Resumo. *Executar testes de vulnerabilidades completos em aplicações Web requer uma sessão autenticada de usuário. Esta autenticação é comumente configurada manualmente por um especialista devido à complexidade dos variados mecanismos de autenticação utilizados por Web apps modernas. Neste artigo apresentamos um arcabouço para descobrir automaticamente as informações necessárias para autenticação programática. Nossa avaliação mostra que o arcabouço utiliza poucos recursos e é eficaz em autenticar em vários Web apps.*

1. Introdução

O número de ciberataques no Brasil aumentou 38% no primeiro trimestre de 2024 em comparação com o mesmo período do ano anterior, sendo os setores mais impactados os de educação, governo, forças armadas e saúde (Leaders 2024). Mitigar ataques de segurança envolve monitoramento contínuo de sistemas expostos à Internet. Atualmente, aplicações Web são uma forma comum de expor serviços cada vez mais críticos a uma quantidade crescente de usuários (Li e Xue 2014).

Porém, a análise de vulnerabilidades em aplicações Web é uma operação ainda majoritariamente manual, que requer o envolvimento de analistas experientes e a utilização de ferramentas de análise de requisições HTTP como o Burp Suite e o ZAP (Zed Attack Proxy). Apesar destas ferramentas automatizarem a execução de testes de vulnerabilidades como *SQL injection* ou *cross-site scripting*, elas requerem configuração complexa e específica para cada aplicação Web alvo (Matti 2021).

Uma etapa crucial para testes de vulnerabilidade com alta cobertura em uma aplicação Web é a autenticação prévia para criar uma sessão de usuário (Jakobsson e Häggström 2022). Testes não autenticados podem falhar em detectar vulnerabilidades se

poucas ou nenhuma funcionalidade da aplicação estiver disponível antes da autenticação; em casos extremos, a aplicação pode exibir apenas um formulário de *login* até que o usuário seja autenticado.

A autenticação em aplicações Web de forma procedural em uma ferramenta de análise é uma tarefa complexa. Por exemplo, a autenticação pode envolver mecanismos contra automação como *captchas* (Al-Fannah 2017) ou requerer múltiplos passos durante a autenticação (Navpreet Kaur 2015).[1] Após a autenticação e abertura de uma sessão de usuário para realização dos testes, uma ferramenta precisa ainda de um mecanismo para verificar se a sessão continua ativa durante os testes, em particular porque os testes podem fazer *logout* (acidental ou propositalmente) e terminar a sessão de autenticação abruptamente. Para contornar estes desafios, as ferramentas de teste possuem mecanismos complexos de controle de sessão que precisam ser configurados, geralmente exigindo envolvimento de um analista (Basso et al. 2010).

Neste trabalho, apresentamos um arcabouço integrado ao ZAP para automatizar a descoberta de informações de autenticação em aplicações Web. Nosso objetivo é facilitar o trabalho dos analistas de segurança e permitir varreduras mais completas em diversas aplicações. O arcabouço identifica o formulário de *login*, incluindo campos de usuário e senha, verifica automaticamente se a autenticação foi realizada e controla a sessão, informando a ferramenta de análise caso ocorra *logout* durante a varredura.

Nossa avaliação mostra que o arcabouço descobre as informações necessárias para configurar corretamente a autenticação no ZAP em aproximadamente 85% das aplicações Web utilizadas para avaliação. A automação da varredura autenticada permite a analistas de segurança focarem na análise dos relatórios gerados pela ferramenta, com impacto direto sobre a segurança (Alazmi e Leon 2022). Nosso arcabouço descobre as informações necessárias para autenticação realizando poucas requisições, permitindo sua aplicação em larga escala. Por último, nossos resultados indicam que varreduras realizadas pelo ZAP com a configuração gerada pelo nosso arcabouço gera resultados equivalentes a varreduras realizadas com uma configuração gerada manualmente por um especialista.

O arcabouço está disponível para a comunidade científica e de analistas de segurança sob uma licença de código livre no GitHub.[2]

2. Fundamentos

Nosso arcabouço, apresentado na seção 3, integra soluções de software livre que descrevemos a seguir. Nos restringimos à utilização de software livre para garantir a máxima aplicabilidade e a capacidade de estendermos o arcabouço futuramente.

Zed Attack Proxy (ZAP). O ZAP é uma ferramenta com várias funcionalidades para testes de segurança em aplicações Web. Ele inclui *scripts* que identificam vulnerabilidades como *SQL injection*, *cross-site scripting*, *cross-site request forgery* e *buffer overflow*. Atuando como um *man-in-the-middle proxy*, o ZAP intercepta requisições enviadas e respostas recebidas de uma aplicação Web. Essas requisições podem ser feitas por um

[1] Por exemplo, o formulário de autenticação em contas da Microsoft apresenta primeiro o campo de email e em uma segunda etapa o campo de senha.

[2] https://github.com/Sacramento-20/Authentication_ZAP_Framework

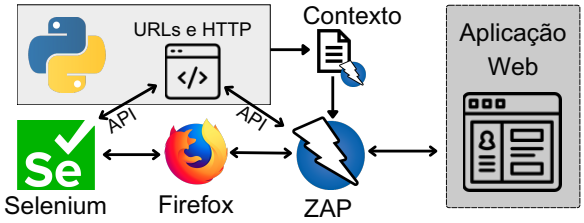

Figura 1. Visão geral do arcabouço na criação do contexto.

navegador durante uma sessão interativa ou pelo motor de varredura do próprio ZAP (Hariyadi e Nastiti 2021).

ZAP possui um Spider para escanear e descobrir URLs em uma aplicação Web, criando uma árvore das URLs visitadas. Ele oferece varreduras ativas e passivas. A varredura passiva faz requisições a URLs diretamente encontradas nas páginas, detectando menos vulnerabilidades, mas sem risco de danos à aplicação (Li e Xue 2014). Já a varredura ativa realiza ataques para identificar vulnerabilidades conhecidas, como *SQL injection* e *cross-site request forgery*, podendo causar modificações no banco de dados (Ashari et al. 2022). É comum usar varreduras passivas em produção e ativas em ambientes de teste. O ZAP classifica as vulnerabilidades encontradas por risco (alto, médio, baixo, informativo) e fornece evidências para cada uma (Tedyyana et al. 2023).

O *contexto* define como varreduras e testes de segurança em uma aplicação Web alvo devem ser realizados. Ele especifica as informações necessárias para o ZAP autenticar na aplicação Web e executar testes de vulnerabilidade, como a URL da aplicação, URL de *login* e *logout*, credenciais de teste, método de controle de sessão e identificação do estado da sessão.

Selenium. O Selenium é um WebDriver que controla navegadores Web. Ele é usado em nosso arcabouço para acessar URLs de login e capturar elementos da página, como formulários e campos de entrada (*e.g.,* nomes de usuário e senha). O Selenium pode ser integrado ao ZAP configurando o acesso à aplicação Web através do *proxy* do ZAP, permitindo que o Selenium inspecione a página e o ZAP analise as requisições e respostas.

3. Arcabouço para Descobrimento de Informações de Autenticação

O objetivo do nosso arcabouço é construir um contexto do ZAP, incluindo as informações sobre a autenticação e controle de sessão do usuário durante testes de vulnerabilidade. O contexto deve ser construído recebendo como entrada apenas a URL da aplicação Web e as credenciais do usuário, descobrindo automaticamente a URL de *login*, o formato da requisição de autenticação e o mecanismo de gerenciamento de sessão.

A Figura 1 apresenta uma visão geral do arcabouço e das ferramentas usadas. Implementado em Python de forma modular, o arcabouço facilita futuras extensões e atualizações. Ele inicializa o Selenium e o ZAP, comunicando-se com eles via suas APIs para acessar o conteúdo das páginas Web e das requisições e respostas trocadas. O Selenium é configurado com o navegador Firefox e o ZAP como proxy. O arcabouço pode ser executado via linha de comando, minimizando a interação dos analistas e permitindo integração com outros arcabouços ou *scripts*. A saída principal é um arquivo contendo o contexto do ZAP para autenticação programática.

3.1. Identificando a URL de *login* e elementos de autenticação

A primeira etapa do nosso arcabouço é identificar a URL de *login* da aplicação. Para isso executamos uma varredura passiva do ZAP para elencar URLs acessíveis a partir da página inicial da aplicação. Para cada URL identificada, utilizamos o Selenium para elencar possíveis elementos de autenticação. Ordenamos a busca nas URLs priorizando aquelas cuja URL contenha algum termo que indique que a URL está relacionada a autenticação ("auth" ou "login"). Esta etapa pode ser ignorada caso o analista informe a URL de *login* diretamente em vez da URL inicial da aplicação Web.

Para cada página, analisamos seu conteúdo e recuperamos todos os elementos da página cujos atributos estão possivelmente relacionados a autenticação. Qualquer elemento com algum atributo name, class, id, type e placeholder cujo valor esteja presente em uma lista de palavras chaves é classificado como um elemento *candidato* (Rusdiansyah et al. 2024). Os elementos candidatos são identificados através de duas listas diferentes de palavras-chave estruturadas como expressões regulares, que os relacionam ao identificador do usuário (`\b[\w.-]user[\w.-]\b` e `\b[\w.-]email[\w.-]\b`) ou senha (`\b[\w.-]password[\w.-]\b` e `\b[\w.-]senha[\w.-]\b`). Dessa forma, o arcabouço criado proporciona robustez na identificação dos elementos no formulário de autenticação, uma vez que as expressões regulares utilizadas podem ser tratadas como substrings em uma palavra, aumentando a probabilidade de captura dos elementos corretos.

Nosso arcabouço tenta realizar a autenticação utilizando todas as combinações de pares possíveis entre elementos candidatos relacionados ao identificador do usuário e de senha identificados na etapa anterior, ordenando os pares de acordo com a ordem de prioridade das palavras chaves que identificaram cada elemento candidato.

3.2. Verificação da autenticação

Para cada tentativa de autenticação da etapa anterior é necessário verificar se a autenticação foi realizada com sucesso. Utilizamos o detector de autenticação embutido no ZAP, que gera um alerta quando a autenticação é realizada com sucesso. Complementamos o detector do ZAP com uma heurística adicional. Utilizamos o Selenium para capturar a URL da *página final* acessada após a tentativa de autenticação (incluindo quaisquer redirecionamentos) e utilizamos o ZAP para obter o conteúdo da resposta da requisição de autenticação e o conteúdo da página final identificada pelo Selenium.

Caso a resposta da requisição de autenticação seja um JSON, aplicamos um conjunto de palavras chave para tentar identificar um token de autenticação. Caso contrário buscamos nos cabeçalhos da resposta algum *cookie* ou algum cabeçalho de autenticação (*e.g.,* "SessionID"). Se ambos estes mecanismos falharem, comparamos o conteúdo da página final acessada após a requisição de autenticação com o conteúdo da página de *login*. Fazemos uma decomposição do HTML das páginas utilizando a biblioteca BeaultifulSoup, calculamos a diferença entre seus elementos e identificamos se novos elementos da página final indicam o estabelecimento de uma sessão autenticada (*e.g.,* "Olá . . . " ou "My Account"). Esta verificação é controlada por uma lista de expressões regulares.

Quando o arcabouço identifica que uma requisição realiza a autenticação do usuário com sucesso, ele salva os elementos da página utilizados como o identificador do usuário e a senha (seção 3.1) mais a URL de *login* no contexto da aplicação. Também averiguamos o conteúdo da *requisição* de autenticação para identificar o mecanismo

de autenticação. O ZAP possui suporte a diversos mecanismos de autenticação; nosso arcabouço suporta os três mais frequentemente utilizados para autenticação programática: formulário, JSON e HTTP. Após identificar o mecanismo de autenticação, o arcabouço salva um modelo (*template*) no contexto para o ZAP construir requisições de autenticação durante varreduras. Por exemplo, construímos um modelo JSON de autenticação substituindo os valores do identificador do usuário e sua senha por %username% e %password%, respectivamente, como esperado pelo ZAP. Por último, o arcabouço salva as informações necessárias para o ZAP verificar se a sessão continua ativa durante a realização de testes de vulnerabilidade. Mais especificamente, o arcabouço salva os cabeçalhos e *tokens* de autenticação que devem ser verificados nas requisições e respostas recebidos do servidor, conforme identificados na verificação da autenticação.

4. Avaliação

Testamos nosso arcabouço em 8 aplicações, 6 aplicações de teste frequentemente utilizadas para teste com o ZAP e duas aplicações reais. Nosso arcabouço conseguiu gerar contextos funcionais para 7 das 8 aplicações.[3] Para validar nosso arcabouço realizamos baterias de testes mais extensas em 3 aplicações de teste. Todos os testes foram realizados em instâncias locais das aplicações executando em contêineres Docker em uma estação de trabalho Ryzen 5 5600G com 32 GiB de RAM.

4.1. Desempenho do arcabouço

Levando em consideração um cenário onde analistas precisam analisar várias aplicações, é importante que o arcabouço descubra as informações de *login* rapidamente e de forma eficiente. Analisamos a quantidade de requisições enviadas pelo arcabouço durante o processo de criação do contexto considerando que a URL da página de *login* é informada ao arcabouço (desta forma a busca pela página de *login* não precisa ser realizada). Encontramos que para as aplicações DVWA, Bodgeit e bWAPP o arcabouço realiza 4, 6 e 6 requisições HTTP para criação do contexto, respectivamente. Todas as execuções completam em menos de 25 segundos, incluindo o tempo de inicialização do Selenium, Firefox e ZAP. Estes resultados indicam que o processo de descobrimento das informações é eficiente, não utilizando muitos recursos computacionais da máquina executando o arcabouço nem das aplicações Web alvo.

4.2. Validação da autenticação

Para validar nosso arcabouço realizamos baterias de testes comparando resultados de varreduras e testes de vulnerabilidade realizados pelo ZAP nas três aplicações alvo. Consideramos três configurações do ZAP: sem autenticação; com autenticação utilizando um contexto gerado pelo nosso arcabouço, denotada *automática*; e com autenticação utilizando um contexto gerado por um especialista, denotada *manual*.

Um desafio que encontramos foi uma alta variabilidade intrínseca nos resultados dos testes executados pelo ZAP, que podem dar resultados diferentes em função de fatores como tempo de resposta do servidor e ordem na qual as requisições são respondidas

[3]A aplicação onde o arcabouço não conseguiu gerar o contexto para autenticação é implementada em Flutter (https://flutter.dev), cujas páginas são geradas dinamicamente por Javascript, o que pretendemos tratar futuramente fazendo uso mais sofisticado da API do Selenium.

Tabela 1. Número de URLs e vulnerabilidades encontradas nas baterias de testes com o ZAP utilizando três configurações de autenticação em três aplicações Web.

Aplicação		DVWA			Bodgeit			bWAPP		
Autenticação		Auto	Manual	Sem	Auto	Manual	Sem	Auto	Manual	Sem
	URLs	129,8	129,0	118,7	101,3	100,1	81,0	45,9	47,0	7,0
Risco	High	11,7	12,4	0,0	5,4	6,0	3,7	1,8	1,8	0,0
	Medium	71,9	68,9	9,0	573,2	572,6	350,9	169,4	166,1	136,0
	Low	123,5	120,4	24,0	136,5	138,0	133,0	207,2	201,7	170,0

(Albahar et al. 2022). Para contornar este problema, executamos o ZAP 10 vezes para cada configuração em cada aplicação alvo e reportamos a média dos valores observados.

A linha "URLs" na **??** mostra o número médio de URLs encontrados em cada aplicação para cada configuração (Mu'min et al. 2022). Ao realizar e validar a autenticação na aplicação no momento dos testes, foi encontrado um número maior de URLs em seu spider, especialmente na aplicação bWAPP, cuja página inicial é a página de *login* e nenhum conteúdo pode ser acessado sem autenticação. Notamos também que a quantidade de URLs encontradas pela configuração automática é equivalente ao número de URLs e vulnerabilidades encontradas pela configuração manual, indicando que a autenticação realizada com o contexto gerado pelo arcabouço desenvolvido é efetiva.

5. Conclusão e trabalhos futuros

Neste trabalho apresentamos um arcabouço para descobrimento de informações para autenticação programática em ferramentas de análise de vulnerabilidades em aplicações web. Nosso arcabouço se mostrou compatível com os mecanismos básicos de autenticação (requisições de autenticação baseadas em formulário e JSON) e, em testes realizados, foi bastante eficaz na seleção dos elementos em formulários de autenticação em aplicações web modernas que utilizam JavaScript. No entanto, possuem algumas limitações. Como o Selenium é utilizado como meio para obter os elementos dos formulários da página, em aplicações, que possui métodos de renderização de DOM dinâmicos, esses campos não podem ser obtidos. Nossa avaliação mostra que o arcabouço é eficiente e efetivo apesar das limitações citadas. Acreditamos que o arcabouço proposto pode ser uma ferramenta útil para times de segurança que gerenciam grandes parques computacionais com muitas aplicações ou com aplicações dinâmicas, como equipes de segurança de universidades e grandes empresas.

Como trabalhos futuros, planejamos avaliar nosso arcabouço em mais aplicações e estender o suporte a páginas que geram o formulário de autenticação dinamicamente (*e.g., single-page apps*). Utilizaremos uma abordagem de envio de requisições pelo ZAP e análise da resposta para verificar a autenticação.

Agradecimentos

Agradecemos aos revisores da SBSeg pelos comentários. Este trabalho foi financiado pela RNP (Hackers do Bem), FAPEMIG, FAPESP, CNPq e CAPES.

Referências

Al-Fannah, Nasser Mohammed (2017). "Making defeating CAPTCHAs harder for bots". Em: *2017 Computing Conference*. DOI: 10.1109/SAI.2017.8252183.

Alazmi, Suliman e Daniel Conte de Leon (2022). "A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners". Em: *IEEE Access*. DOI: 10.1109/ACCESS.2022.3161522.

Albahar, Marwan, Dhoha Alansari e Anca Jurcut (2022). "An Empirical Comparison of Pen-Testing Tools for Detecting Web App Vulnerabilities". Em: *Electronics*. DOI: 10.3390/electronics11192991.

Ashari, Ilham Firman, Vina Oktariana, Ringgo Galih Sadewo e Salman Damanhuri (2022). "Analysis of Cross Site Request Forgery (CSRF) Attacks on West Lampung Regency Websites Using OWASP ZAP Tools". Em: *Jurnal Sistem Informasi dan Teknologi*. DOI: 10.32736/sisfokom.v11i2.1393.

Basso, Tania, R Moraes e M Jino (2010). "Uma abordagem para avaliação da eficàcia de scanners de vulnerabilidades em aplicações Web". Em: *Campinas, Brazil: Dissertação de Mestrado. Universidade Estadual de Campinas*. DOI: 10.47749/t/unicamp.2010.772795.

Hariyadi, Dedy e Faulinda Ely Nastiti (2021). "Analisis Keamanan Sistem Informasi Menggunakan Sudomy dan OWASP ZAP di Universitas Duta Bangsa Surakarta". Em: *Jurnal Komtika (Komputasi dan Informatika)*. DOI: 10.31603/komtika.v5i1.5134.

Jakobsson, Adam e Isak Häggström (2022). *Study of the techniques used by OWASP ZAP for analysis of vulnerabilities in web applications*. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-186346.

Leaders, Security (2024). *Ciberataques aumentam 38% no Brasil durante primeiro trimestre de 2024*. URL: https://securityleaders.com.br/ciberataques-aumentam-38-no-brasil-durante-primeiro-trimestre-de-2024/.

Li, Xiaowei e Yuan Xue (2014). "A survey on server-side approaches to securing web applications". Em: *ACM Comput. Surv.* DOI: 10.1145/2541315.

Matti, Erik (2021). *Evaluation of open source web vulnerability scanners and their techniques used to find SQL injection and cross-site scripting vulnerabilities*. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-177606.

Mu'min, Muh. Amirul, Abdul Fadlil e Imam Riadi (2022). "Analisis Keamanan Sistem Informasi Akademik Menggunakan Open Web Application Security Project Framework". Em: *Jurnal Manajemen Informatika*. DOI: 10.30865/mib.v6i3.4099.

Navpreet Kaur, Mandeep Devgan (2015). "A Comparative Analysis of Various Multistep Login Authentication Mechanisms". Em: *International Journal of Computer Applications*. DOI: 10.5120/ijca2015906472.

Rusdiansyah, Rusdiansyah, Nining Suharyanti, Hendra Supendar e Tuslaela Tuslaela (2024). "Web Program Testing Using Selenium Python: Best Practices and Effective Approaches". Em: *Sinkron: jurnal dan penelitian teknik informatika*. DOI: 10.33395/sinkron.v8i2.13569.

Tedyyana, Agus, Osman Ghazali e Onno W Purbo (2023). "A real-time hypertext transfer protocol intrusion detection system on web server". Em: *TELKOMNIKA Telecommunication Computing Electronics and Control*. DOI: 10.12928/telkomnika.v21i3.24938.