Utilizing Dynamic Analysis for Web Application Penetration Testing

1st Patrik Pis
Department of Telecommunications
FEEC, Brno University of Technology
Brno, Czechia
xpispa00@vut.cz

2nd Willi Lazarov
Department of Telecommunications
FEEC, Brno University of Technology
Brno, Czechia
lazarov@vut.cz

Abstract—This paper presents the design and implementation of a new modular tool, called PtWebDA, for dynamic analysis of web applications as one of the techniques used in penetration testing. Compared to other available tools and their limitations, our solution enables efficient rate limiting while also allowing testing of HTTP headers, cookie attributes, and content security policy directives. To verify its effectiveness in supporting manual web application penetration testing, we performed experimental testing in a controlled environment. The results of testing the presented tool PtWebDA are discussed in detail and highlight the key contributions of our solution.

Index Terms—cybersecurity, dynamic analysis, penetration testing, rate limiting, cookies, CSP directives, HTTP headers

I. INTRODUCTION

Information security practice, despite a gradual increase in awareness, is still lagging behind the rapid development of complex applications and systems. Security testing is still not a standard procedure during application development. The gradual increase of complexity of applications results in increasing number of opportunities for new errors and vulnerabilities. Most modern companies present themselves to the public using websites, and internal business processes and their management are gradually moving to the cloud in the style of complex web applications and services. The focus on secure development and operation of web services is thus becoming increasingly desired. With threats on the rise, testing web application security is becoming essential for secure operation of web services and reduction of potential negative impact on customers, employees, or the entire company.

Web applications have become a popular way for sharing content with users or providing services in various industries such as online banking, e-commerce, social networks, and many others. Web applications make it easier for users to access services without the need to install proprietary software, and thus become the easiest way for companies to provide content or service to a wide range of customers. However, due to their accessibility to the public, web applications are not only exposed to potential customers, but also to potential attackers who can cause their unavailability. For this reason, the overall security of web applications is critically important.

The research described in this paper was financially supported by the Ministry of the Interior of the Czech Republic, project No. VK01030019.

The impact of cyberattacks on web applications can vary widely and is largely influenced by the technologies used and the nature of the web application itself. However, if vulnerabilities found in a web application are exploited, not only will the confidentiality, integrity, or availability of the web application operator’s assets and infrastructure compromised, but sensitive data belonging to users may also be leaked, which can have an even deeper impact [1].

The main contribution of this paper is the design and development of a new tool PtWebDA for dynamic analysis of web applications that aims to support manual penetration testing. PtWebDA can test rate limiting, HTTP headers, cookie attributes, and Content Security Policy (CSP) directives.

The paper is divided into 5 sections. Section II discusses background and related work. Section III focuses on the description of the proposed dynamic analysis tool, including its key features and implementation details. Section IV presents the experimental testing of the tool developed, including a discussion of the results. The last Section V highlights the presented solution and indicates possible future work.

II. BACKGROUND AND RELATED WORK

Web applications are constantly evolving, and it is extremely important to pay adequate attention to the study of their security, as the increase in the complexity of web services can lead to new vulnerabilities, new tactics of threat actors, or new methods of circumventing the already existing defense mechanisms of web applications. To assess the security of a web application, it is crucial to point out the most critical vulnerabilities that are present in modern web applications. The OWASP Top 10 is a document of 10 categories that lists the most common vulnerabilities present in web applications. This document is put together and maintained by an organization called the Open Web Application Security Project (OWASP) [2]. OWASP is a non-profit organization dedicated to improving the security of web applications. The OWASP Top 10 is updated regularly to reflect current trends and threats in application security. It includes various types of vulnerabilities (e.g., broken access control) or security misconfigurations, from authentication and authorization flaws to vulnerabilities caused by incorrect processing of user input and exploitation of publicly known vulnerabilities.
One of the methods that we can utilize to assess the security of web applications is dynamic analysis. Dynamic analysis can be understood as testing or evaluating the functionality or security of an application during its runtime. It can be utilized in a wide array of scenarios, as it is non-exclusive to security testing. Such scenarios include software quality testing [3] and malware analysis [4]. This type of analysis also requires active execution of the tested application. During dynamic analysis, the individual tools do not have access to the source code as in static analysis, which does not require the web application to run, and its process and techniques are thus different from dynamic analysis. These tools simulate end users and have the same access to the application’s sources as its potential users and their devices [5], [6].

Dynamic analysis of web applications can be performed manually or automatically. Manual and automated analysis of web applications is applicable in many test scenarios, but the choice of the right approach is strongly dependent on the nature and specification of the test. However, both approaches require a black-box approach without any insight into the internal structure of the web application. Dynamic analysis and penetration testing are two very closely related concepts. Principles of black-box penetration testing of web applications imply that dynamic analysis is a kind of building block of the whole security assessment. To perform a black-box penetration test, a given web application must be running, either in a test or production environment and the tester examines its reactions to various stimuli in real time [7].

To perform dynamic analysis, there already exist publicly available open source or paid commercial tools. Although the terms dynamic analysis, dynamic application security testing (DAST), and vulnerability scanning are different in definition, in practice these terms are often used interchangeably. In the context of the presented solution, we therefore focus on tools for dynamic analysis of web applications. The comparison of selected tools is shown in Table I.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Fuzzing</th>
<th>Manual</th>
<th>Automated</th>
<th>Open source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burp Suite [8]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>OWASP ZAP [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ffuf [10]</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Invicti [11]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Acunetix [12]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>w3af [13]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Nikto [14]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AppSpider [15]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>WebInspect [16]</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

III. IMPLEMENTATION OF TOOL FOR DYNAMIC ANALYSIS

As shown in Table I, many publicly available tools lack support for a manual approach to testing. The biggest problem associated with automated dynamic analysis is the fact that in the current state-of-the-art, it is not capable of detecting all vulnerabilities. Certain more complex vulnerabilities, or chains of vulnerabilities, still require human thinking and creativity. Although automated dynamic analysis is a powerful tool for security testing, the manual approach is still more successful in finding vulnerabilities and eliminating false positives. The tool we are proposing in this paper should serve as the best of both worlds. The tool still automates the important tasks, but the tester has full access to the process and can adjust the testing parameters based on the observed behavior.

Compared to other tools, PtWebDA only tests for a limited set of security issues. This is done by design and should be considered only as an addition to a penetration tester’s toolkit during a manual penetration testing engagement, not as a standalone automated security testing solution. The high-level diagram of PtWebDA is shown in Figure 1.

![High-level diagram of PtWebDA](image)

To show the difference and key contributions of our solution, we compared PtWebDA with already mentioned tools, focusing mainly on implemented modules for dynamic analysis. The results of the comparison are listed in Table II, where the rate-limiting test does not include any of the tools compared. This result was expected given that most automated tools care about covering as much as possible predefined tests and not the operation security.

1 Secure testing, i.e. limiting the speed of tools or invasive tests in order to avoid detection of the activity in question.
The tool was developed in Python programming language and, as of the time of writing this paper, consists of four modules representing the four aforementioned tests: HTTP headers (general), cookie attributes, CSP directives, and rate limiting, which are described in the following sections.

A. HTTP Headers

Testing for the presence of HTTP headers involves capturing the HTTP response to a request to a web server. Using the Python requests library [17], the test executes an HTTP request with a GET method to the specified URL that was passed to the test as a parameter. While this test is inherently very simple and easy to implement, the power of the findings it can provide can have a significant impact on the overall security of a web application or web server. Additionally, HTTP headers can often be missed during manual inspection of responses, and the tool helps to highlight potential issues to the tester.

B. Cookie Attributes

The principle behind cookie attributes and CSP directives testing is practically the same as in HTTP headers involving capturing the HTTP responses to requests to a web server. However, this time the tool focuses more on the contents of these HTTP headers than on their presence alone.

The cookie attributes module in the current version of PtWebDA focuses mainly on these cookie attributes: Secure, HttpOnly, and SameSite. These attributes play a crucial part in lowering the potential impact of vulnerabilities such as cross-site scripting (XSS) or cross-site request forgery (CSRF), since they directly contribute to the definition of how and when entities may access the cookies themselves. The module checks these attributes, analyzes them, and informs the tester of possible misconfigurations.

C. CSP Directives

Content Security Policy (CSP) defines the rules on how a web browser should load and execute content provided by the web application. It requires a careful setup and precise definition. If CSP is defined, it has a significant impact on the way web browsers render web pages. CSP helps detect and prevent a wide range of attacks, including cross-site scripting, other cross-site attacks, and other attack vectors that lead to compromise users’ privacy.

This module analyzes the CSP directives defined in the Content-Security-Policy HTTP header and searches for insecure definitions and misconfigurations. The important thing during CSP analysis is that there are multiple ways to execute client-side JavaScript, not just direct code definitions (e.g. in <script> tags). JavaScript code can be executed from CSS style definitions, which are often neglected during CSP configuration and may lead to data exfiltration. This module focuses on informing the tester about these possible misconfigurations and highlights possible dangerous CSP directives.

D. Rate Limiting

Testing rate limiting in dynamic analysis involves building a model to evaluate the response of a web application to a significant number of requests in a specified amount of time. However, the actual fact that a web application or a web server has a mechanism for limiting user requests may not be immediately obvious. It is a significant challenge for a tool evaluating rate limiting to differentiate between the slowdown or complete termination of requests by the actual rate limiting mechanism and the insufficient computational or memory resources of the tested system.

Mathematically, we can describe the rate limiting with the following equation:

\[ \int_{t_1}^{t_2} R(t) \, dt \leq L \times (t_2 - t_1), \quad (1) \]

where

- \( R(t) \) represents the rate of HTTP responses in time \( t \),
- \( L \) represents the maximal number of requests per minute,
- \( t_1, t_2 \) represent the time interval.

Analysis of best practices from companies such as CloudFare [18] shows that relatively low values of the maximum number of requests per specified time are used for the effective application of rate limiting. Thus, an effective defense against attacks on web application login forms can be a rate limit of, for example, 4–10 requests per minute. PtWebDA performs an analysis of this HTTP traffic and determines the number of requests necessary to trigger the rate limiting mechanism. The detection of rate limiting and the number of failed and successful requests provide sufficient information to solve the inequality defined in Eq. 1 and thus allow us to compute the theoretical value of the maximum number of requests per minute for a given time interval.

From the penetration tester’s perspective, this provides important information on the target’s security status and shows how testers can alter her/his approach by trying to circumvent the rate limit, e.g. by limiting the number of requests just below the detection threshold, allowing the tester to continue with testing uninterrupted. In addition, the tool does not perform automatic scans without the tester’s knowledge, which makes it suitable for manual penetration testing, where the tester has the entire tool under control.
IV. Experimental Testing

As described in section III, the tool consists of four working modules in its first version. We conducted a series of tests to prove the effectiveness and reliability of each module before testing it out in the production environment. To test it, we created a controlled environment using virtual machines. The virtual machine was running on the Ubuntu 22.04.3 LTS operating system and used Apache 2.4.58 as its web server. To create a controlled sandbox environment, we created a misconfigured web application written in Python using a Flask web framework, containing all the flaws the tool is expected to successfully detect. The web application consisted of a total of three endpoints; ‘/login’, ‘/menu’, and ‘/nolimit’.

The login endpoint had a rate limit configured as low as 4 requests per minute, the menu endpoint had a rate limit configured as 30 requests per minute, and the final endpoint had no rate limit configured. At the same time, the different endpoints of the web application returned the content with different set of HTTP headers and their contents. For example, the login endpoint did not return the Content-Security-Policy header, but the menu endpoint did. A summary of results of the experimental testing conducted in our controlled environment is shown in Table III.

### Table III: Results of Experimental Testing

<table>
<thead>
<tr>
<th>Tested module</th>
<th>Findings (found/total)</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP headers</td>
<td>8/8</td>
<td>&lt; 1s</td>
</tr>
<tr>
<td>Cookie attributes</td>
<td>3/3</td>
<td>&lt; 1s</td>
</tr>
<tr>
<td>CSP directives</td>
<td>4/4</td>
<td>&lt; 1s</td>
</tr>
<tr>
<td>Rate limit</td>
<td>1/1</td>
<td>18.46s</td>
</tr>
</tbody>
</table>

The main goal of the experimental testing was to ensure that the proposed tool was capable of detecting and effectively highlighting important insights in web server and web application configurations and to try to identify potential bottlenecks and areas for future improvement.

The controlled environment provided us with a strong position to interpret the output of individual module’s test runs and compare them with the intended misconfigurations. Since all modules, except rate limiting, are very basic in their core principles, the results of the experimental testing were very accurate. The tool was effectively able to identify all missing headers and provided output regarding headers that reveal information. Regarding the rate limiting module, the results were accurate as well with a few exceptions. The rate limiting module uses multithreaded approach and can estimate the web application’s rate limit settings with an accuracy of approx. 5 requests margin. The experimental testing of rate limiting provided us with insights on how to handle some edge cases, which can alter the tool’s output and produce false information. Some of the edge cases include; the tool is too fast and the rate limit is quite small, or the rate limit is very permissive and the tool is not fast enough to trigger the rate limit, or the tool does not send enough requests to trigger the rate limit defined by the tested application.

V. Conclusion and Future Work

To summarize our paper, we reviewed the current state of dynamic analysis with a primary focus on web application penetration testing, including a comparison of available solutions. Based on the limitations of existing tools, we designed and developed a new modular tool, PtWebDA, to test the rate limiting, HTTP headers, cookie attributes, and CSP directives of web applications. The tool is developed to be fully under the control of the tester, which makes it particularly suitable for manual penetration testing.

Based on the experimental testing, the tool can be improved in its efficiency and reliability. Additionally, dynamic analysis of web applications is not limited to rate limiting, HTTP headers, cookie attributes, and CSP directives. The area of dynamic analysis is wide, and the modular design of the PtWebDA allows us to develop it even further and transform it into a more complex and versatile tool. Future work and development of this tool will include the development of more modules covering more testing scenarios included in the OWASP Top 10 methodology.

REFERENCES