Vulnerability Scanner: Web-based Security Testing

Andrei-Daniel ANDRONESCU, Ioana-Ilona BRĂSLAȘU, Dumitru-Iulian NĂSTAC
Faculty of Electronics, Telecommunications and Information Technology,
University POLITEHNICA of Bucharest, Romania
andronescu.andreidaniel@gmail.com, ioanabraslasu2000@gmail.com, iulian.nastac@upb.ro

Abstract

As the use of internet-based software increased, cybersecurity has emerged as a major issue in the current world. The fast-paced technology innovations allowed most companies to scale their business, consumers to access easier their favorite products, thus increasing the reliance on web-based software. The importance of web security cannot be emphasized given the increase in cybercrime and the damage it poses to businesses, people, and governments. This paper proposes an automated solution capable of detecting and exploiting common vulnerabilities found on web-based software, this being done without performing any malicious intended operations. By using software capable of automatically detecting the means a client could communicate with a server, users can ensure that a thorough verification is done on their web-applications, revealing the blind spots that developers may have overlook.

Index terms: Chromium, File Inclusion Attacks, NodeJS, Puppeteer, SQL injection, vulnerability scanner, web application security, testing

1. Introduction

In today's age, the internet is responsible of most business operations. It provides the companies the ability of communicating and collaborating with customers and partners instantly. It offers the companies the access to a huge amount of data, that, if understood correctly, can improve its revenue and its overall performance. As a result, every modern company is in some shape or form, depended on the internet. But, as the saying goes, "with great power, comes great responsibility". The responsibility in their case is ensuring the security of the company's data as well as customer's sensitive data. One simple easily accessible vulnerability in the system could cause the loss of customer's trust, the loss of competitive edge or even the company itself.

Malicious actors are always seeking for those opportunities. It is well known that there are armies of bots (botnets [1]), constantly scanning the internet for newly deployed servers. Everything that you expose on the internet should be protected, and security measures must be taken way before the service becomes public. Otherwise, malicious actors can easily plant malicious content on your server, plant backdoors allowing them later access, thus exposing your own internal network.

It's always easier to prevent, than treat. Ensuring that a website is secure before making it public can save you from lots of unwanted attention. Therefore, in our paper, we're trying to propose a testing software, capable of detecting common attacks, that bots or even real people will use. Those common attacks consist of SQL injections, cross-site scripting, and buffer overflows.

2. Design

Generally, websites offer clients the possibility of interacting with the server through login forms, search forms, etc. This, in turn, allows the users to have access (although limited) to the back-end systems of a website through specific APIs or HTTP requests. Web applications are becoming more and more of a popular target for attackers, and they will try to exploit the previously mentioned methods in order to gain information and control. Ensuring that web applications are secure is a critical step that needs to be taken by organizations to keep their client's trust. One way that this security can be ensured is by using a vulnerability scanner that can identify potential safety issues in web applications.

In this paper, we propose one such vulnerability scanner that is capable of detecting potential security flaws in web-based applications. Our scanner utilizes a five-step process, which includes a website crawling module, the processing of input forms, a malicious payload generator, a payload injection phase and response analysis. The diagram of the system can be observed in Fig 1.

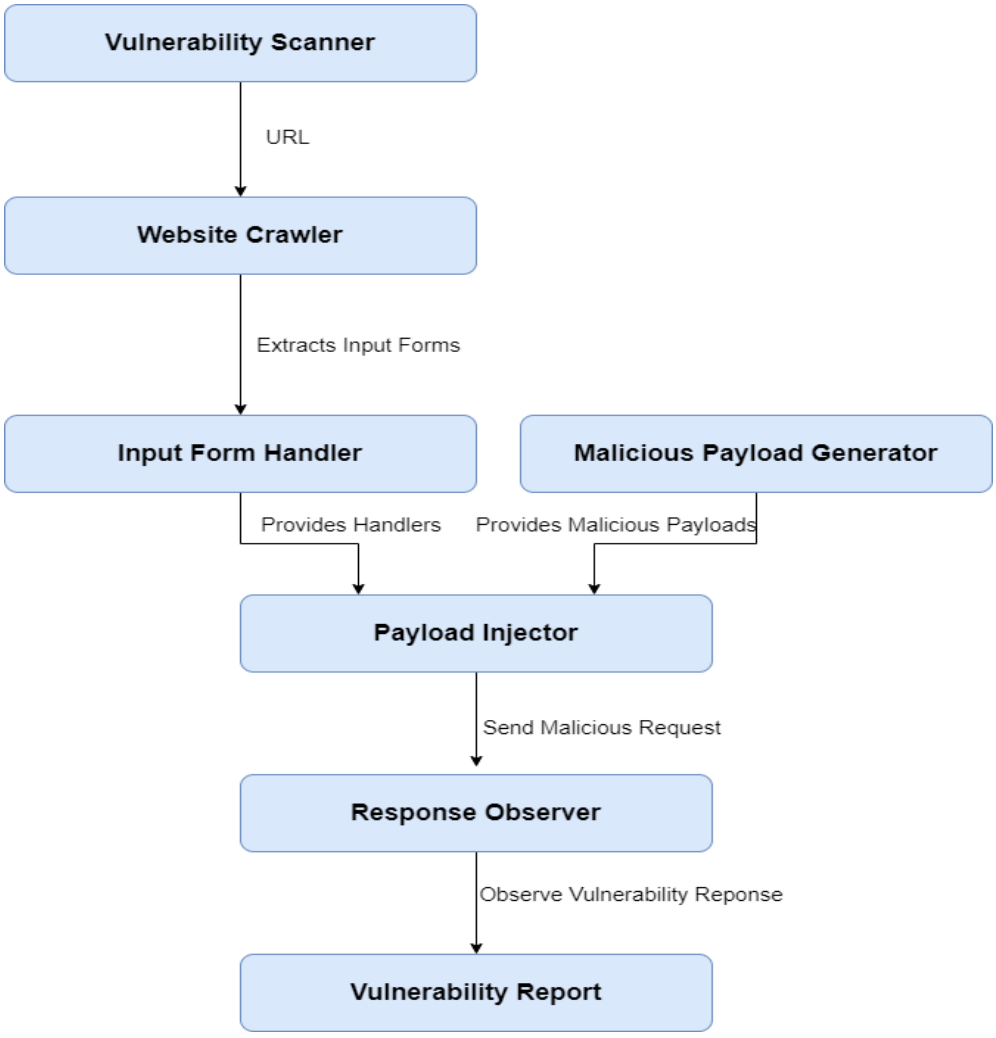

Fig. 1. Vulnerability Scanner diagram for a single web page

2.1. Web Crawling Module

In this article, we propose an exhaustive method of scanning the vulnerabilities of a web server. For this reason, it is important to be able to verify each easily accessible communication bridge between client and server. Thus, a web crawler is used in order to extract all the input forms available on the specified web page. It's also important to learn that a web domain might contain many resources, and attackers always try to find the weak link in a domain to gain access. One old server

running on an outdated version of Apache, one server running on a Java backend taking use of an outdated Log4j module [2] will be the primary target of an attack. Therefore, a scanning process is necessary when using a vulnerability scanner. Our implementation of resource scanning is performed not only on the given web page, but rather in a recursive manner. The web page represents only the starting point in the reconnaissance mission. Links to other resources are scraped out of each web page, and if the link is part of the same domain, it will be followed.

It is well known that modern websites contain some sort of protection against automated software that try to access it (Captcha2, ads, honeypots, automated bot prevention solutions, hidden fields, IP monitors). Bypassing those security measures is a complex task, and it does not represent the scope of our paper. However, in the development process of the web crawler, it was observed that a good website accessing rate can be achieved by removing ads present on most web sites, by using an Ad Blocking browser extension. One other important factor that impacted the accessing rate of websites using automated software was a security feature hosted on the websites that inspects the client's web browser profile. To mitigate this issue, a real web browser profile was used, profile that was manually created and exported for later use in each crawling process.

The crawling process was built on top of the Chromium web browser. A Chromium instance was launched by the NodeJS back-end and controlled with the help of the *puppeteer* module in order to simulate human-like behavior. By using this configuration, we ensured that we had access to a broader range of websites, mostly because it supports scraping from dynamic web pages. Once accessing a web page, the extraction process occurs by identifying the elements of interest such as input forms and links with the use of selectors. Links are stored for later use, when the testing process is fulfilled for the current web page. From the input forms, event handlers are extracted, handlers that can be used to populate the input fields, with the specified malicious payload and to submit those payloads.

2.2. Malicious Payload Generator

The Malicious Payload Generator represents a separate, highly scalable, part our system. It is critical component in our Vulnerability Scanner, being responsible of generating HTTP-based attack payloads. The malicious load must be carefully designed when used against real life websites. Only payloads that can validate the presence of a vulnerability in a system must be used in this case such that no malicious actions are performed. This type of attacks is also known as "nospoilt".

This module generates a variety of attacks which are determined by the web application under scan and the type of input fields. Numerous attacks can be produced, including the following most typical ones:

- **SQL Injection Attacks**. Significant data breaches and confidential information loss can result from those types of attacks [3]. They mostly consist in the injection of SQL code in the input fields that interact with a database. Those codes are then executed as an SQL query and the result returned to the malicious actor. This allows him to gain access to secret information such as user credentials, to update the prices on an e-commerce website, or to obtain Business Intelligence regarding the company's financial situation. This is why, servers must parse queries sent by the users, in order to be sure that those type of attacks cannot be executed.
- **XSS attacks**. Modern web servers are using dynamic web pages, using JavaScript that provides most of the robust functionality [4]. XSS attacks (also known as cross-site scripting attacks) are executed when unsuspecting users access the affected page. The attack consists of injection of malicious script in a victim's browser [5], resulting in the tracking of user's activity, or performing other malicious tasks.

- **Command Injection Attacks**. The input fields are the primary targets for those type of attacks. They inject malicious commands that the server executes due to the insufficient input validation [6], using the privileges of the web application. A successful attack can lead to a full system compromise and data leaks, thus posing serious threats to a web application.
- **File Inclusion Attacks**. Local File Inclusion (LFI) attacks generally occur when a web server allows users to access files on a web server. Once an attacker gains the means of accessing files on the server directly from the browser [7] (example: finding a php script that can open a file on the system), the system is compromised. Remote file inclusion (RFI) attacks on the other hand, are performed by exploiting web server functions that can reference different resources, outside of the domain. This allows attackers to upload malware on the server and generally result in information theft and site takeovers [8].

2.3. Payload injector

The payload injector is a module that is responsible for receiving the input handler from the web crawler, identify the type of input type being used. With this obtained information, it can perform a request on the Malicious Payload Generator module to retrieve a suitable payload, that can be used upon the handler.

The payload injector sends the HTTP request to the server and awaits for a response. One important thing that must be taken into account is that for a successful and thorough attack performed on the server, the payload injector needs to be find tuned on the specific website, with thorough understanding of the web application under test. Our paper proposes a general approach, more oriented towards a general testing scenario. With this in mind, if the testing of a specific domain needs to be performed, a fine tuning on the payload injector needs to be done. Additionally, it is extremely important to use a custom implementation of the payload injector in a responsible manner, by requiring permission from the owner. In the development process of this project, no servers were used without proper clearance, and with explicit malicious intent.

2.4. Response Observer and Vulnerability Report

The Response observer module is responsible for detecting if an attack was successful. This is done by analyzing the server's response for the HTTP request sent by the Payload Injector. Each type of attack would have its characteristic response type. This is an important challenge to overcome when designing an automated vulnerability scanner, since each type of attack has a different outcome, and even the same attack can have different outcomes when tried against different systems. For this reason, detection heuristics fine-tuned for each attack were added. A general overview of the detection heuristics that can be used for detecting attacks will be presented in the following lines.

In the case of an SQL injection attack, the module can analyze the response data received, and identify if any anomalies are present. It will look for specific signatures that represent a successful SQL attack, such as error messages, unexpected status codes, database information that would contain specific keywords in relation to the provided malicious input. If any of those signals occur, the response observer will treat the SQL injection as successful.

In the case of XSS attacks, it is important to note that automatically detecting if an XSS attack successful is a serious challenge since the behavior of those attacks is different. One common scenario is that the attacker tries to insert a script code in order to get some unauthorized data. This XSS attack is known to be present if an unexpected <script> tag is present in an input location. Another method for checking if the XSS attack is present is by verifying with a network traffic monitor if the requests are sent to a location outside the website's domain.

Detection of a successful command injection attack depends mostly on system that has been targeted and on the type of command itself. Verifications if data has been leaked are observed, or verifications if errors are observed in the case of intentionally erroneous injected commands.

File inclusion attacks are also hard to detect if they were executed successfully. In general, insertion of some malicious files that could trigger to be executed by the server may have some sort of expected behavior which allows us to verify its effectiveness. For example, trying to access the /etc/passwd file in the server's filesystem might return a string containing the word "root" [9], or inserting a php wrapper containing a URL to our own website can be used to detect if the attacks were successful.

3. Conclusion

In conclusion, the automation of identifying vulnerabilities in web applications is studied in this article. This project implied the incorporation of Node.js, Chromium, and Puppeteer to create a modern and capable vulnerability scanner. The scanner extracts input forms and navigates additional links to generate HTTP-based attacks using the Malicious Payload Generator module and then tests their effectiveness with the Payload Injector module. The Response Observer, with its specific criteria for each attack type, determines whether the attacks were successful.

The project automates the testing of web application security to help developers pinpoint possible vulnerabilities and tackle them before hackers can capitalize on them. It is possible to tailor the attacks to fit individual needs and requirements with the tool. Through the automation of testing capabilities, a multitude of vulnerabilities can be detected and addressed early in the software development process. Ultimately, this can help organizations reduce their vulnerability to cyber-attacks and improve the security of their applications.

The efficiency of automated vulnerability testing software is directly proportional to the amount of attack types it can perform, detect and report on. Future work will focus on this idea, of increasing its capabilities in terms of the number of attacks it can perform and refactoring the actual project in order to make it more scalable and capable of bypassing more bot detection mechanisms.

Finally, our research team conducted this study with ethical considerations, and we ensured that no live servers were compromised or damaged in the process of developing this paper.

References

[1]. P. Sabanal, IBM. Thingbots: The Future of Botnets in the Internet of Things. [Online]. Available: https://securityintelligence.com/thingbots-the-future-of-botnets-in-the-internet-of-things/ [Accessed: Apr. 20, 2023].

[2]. National Institute of Standards and Technology - CVE-2021-44228 [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-44228 [Accessed: Apr. 21, 2023].

[3]. Stuard McDonald, SQL Injection: Modes of Attack, Defence, and Why It Matters [Online]. Available: https://www.sans.org/white-papers/23/ [Accessed: Apr: 21, 2023].

[4]. Sucuri. Cross-Site Scripting (XSS) Attacks. [Online]. Available: https://sucuri.net/guides/what-is-cross-site-scripting/ [Accessed. Apr. 21, 2023].

[5]. Kirsten S., Cross Site Scripting (XSS). [Online]. Available: https://owasp.org/www-community/attacks/xss/ [Accessed Apr. 21, 2023].

[6]. Weilin Zhong, OWASP. Command Injection. [Online]. Available: https://owasp.org/www-community/attacks/Command_Injection [Accessed Apr. 21, 2023].

[7]. Admir Dizdar (9 July 2021). LFI Attack: Real Life Attacks and Attack examples. [Online]. Available: brightsec.com/blog/lfi-attack-real-life-attacks-and-attack-examples/ [Accessed Apr. 21, 2023].

[8]. Imperva. Remote file inclusion (RFI). [Online]. Available: https://www.imperva.com /learn/application-security/rfi-remote-file-inclusion/ [Accessed Apr. 21, 2023].

[9]. Local File Inclusion (LFI) – Web Application Penetration Testing. [Online]. Available: https://medium.com/@Aptive/local-file-inclusion-lfi-web-application-penetration-testing-cc9dc8dd3601 [Accessed Apr. 22, 2023].