VulnScanner

A Generic python scanner to detect injection vulnerabilities in Web-applications

Krishna Vamsi G
Computer Science- Fall 2022
University of Central Florida
Florida, USA

Khoushik Reddy C
Computer Science- Fall 2022
University of Central Florida
Florida, US

***Abstract*— In the last few years, there have been a lot of web applications released all over the world. Similarly, cyber-attacks targeting vulnerabilities in web applications are on the rise as well. Consequently, web applications need to be made more secure in such circumstances. The assessment and prevention of vulnerabilities require a deep understanding of them. It is, however, very time-consuming and difficult to manually check every single vulnerability on the web. Therefore, we need a web application vulnerability scanner. The purpose of this paper is to present a Python program that will automate the scanning of web applications for injection vulnerabilities and the locations of those vulnerabilities. We emphasize the importance of identifying cross-site scripting (XSS) and structured query language injection (SQLi) vulnerabilities. To determine the detection rate, we tested the program on different intentionally vulnerable websites.**

Keywords*— *SQL injection, cross site scripting, Web applications, security, python vulnerability scanner, Injection vulnerabilities, XSS.

I. INTRODUCTION

We live in an era where web applications are an integral part of our daily lives. We deal with numerous custom-built web applications utilizing multiple specialized technologies. The problem with many of these applications, however, is that they are associated with vulnerabilities. Web application developers have difficulty keeping up with emerging vulnerabilities and newly discovered attacks because of the complexity of the web, which includes different programming languages, encoding standards, browser clients, and scripting languages and environments.

In the past, it was common for applications to be deployed in closed client-server or stand-alone environments. In the current age, security measures such as network security and firewalls are not adequate to protect against Web-based attacks since these applications are accessible to everyone. In order to assess and prevent vulnerabilities, it is essential to have a thorough understanding of how vulnerabilities work. Nevertheless, manually checking every web vulnerability is a near to impossible task for humans. As a result, we need to have a tool that scans our web applications for vulnerabilities.

A vulnerability scanner for a web application is a scanning tool used by app developers and security personnel to detect web application vulnerabilities that can be exploited. As part of this paper, we emphasize the importance of identifying cross-site scripting (XSS). Additionally, the proposed system can also detect SQL injections to a degree.

An XSS attack involves infiltrating websites with malicious code. By exploiting a flaw in a target web application, an attacker can send malicious code to the target. The XSS attack occurs when an attacker causes a web application to send data in a form that can be interpreted by the user's browser. In the case of XSS attacks, malicious content is delivered instantly or upon loading or performing a specific action. A web application might be particularly vulnerable to XSS attacks because the attackers appear within a trusted domain.

SQL injection (SQLi) allows attackers to interfere with database queries made by web applications. It is generally the case that an attacker can access data that they would not normally be able to see. As a result, other users' data could be accessed by the application or any other data that the application has access to. This data is often modified or deleted by attackers, changing the application's content or behavior persistently.

A note should be made about the importance of integrating security into the software development life cycle in a proactive manner throughout the development process. This is in contrast to adding and testing it after the development cycle has ended.

II. LITERATURE REVIEW

Research and security professionals have been addressing vulnerabilities like XSS and SQLi since these threats first emerged. During the life cycle of security

auditing several strategies are taken into account to reduce the number of vulnerabilities present in an application. These could include penetration testing, static and dynamic analysis of source code, black-box and white-box testing and several other approaches. Each approach produces a unique result and provides the developer with a new perspective. Usually, multiple approaches are followed during the life cycle. Although following multiple approaches can utilize a lot of resources such as time, effort and money, but this methodology makes sure to reduce the overall vulnerabilities present in an application.

Some of the previous works addressing the same issue did not require any details of the target web application as it followed the black box testing strategy and solely detected XSS vulnerabilities [1]. This approach finds all the possible XSS attacks that can be performed on a web application. But this approach does not find vulnerabilities after a user log into the web application.

Our program follows grey box testing as it requires user login credentials and focuses on both XSS and SQLi vulnerabilities. By providing the user credentials the crawler implemented in our code searches for all the links which have forms present in the entire web application. Once these forms are found the program sends the payload into each form individually and checks if the payload works or not. The code is provided with payloads for both XSS and SQLi.

The system we proposed is based on the following main characteristics:

- The scanner does not need access to the source code of the web-application, however login credentials are required by the scanner to identify threats in the application after an end user logs in.
- The scanner is very flexible and can be used to test any website by making minor changes to the scanners code according to the architecture of the web application.
- The proposed system can not only detect XSS attacks but also detect SQL injections up to some extent.

III. Proposed Work

Problem statement: OWASP's Top 10 (2021) lists Injections as the third most prevalent issue, which is present in more than two thirds of all applications. It's estimated that more than 60% of web applications are susceptible to XSS attacks. In 2021, a report produced by OWASP stated that 274,000 occurrences of SQLi attacks were observed on applications that were tested by them.

There are several risks associated with XSS and SQL injection. XSS attacks can be used to allow an attacker to hijack the victim's session and take over their account. Meanwhile SQLi attacks can allow an attacker to read confidential data present in a database, manipulate data/information present within the database or even perform operations on the database with administrative privileges.

In order to avoid these attacks, there are several procedures a developer can follow to ensure that most of the threats present in a web application are taken care of. One of the ways a developer can make sure that their website is not prone to the latest attacks is by regularly scanning their application by using a scanner which provides malicious inputs and records the behavior of the website when it interacts with the input. This can help a developer as it would shed light on what kinds of inputs can exploit the web application and potentially lead to the compromise of the website by an attacker. If the developer is aware of such kinds of inputs, these vulnerabilities can be taken care of long before someone can exploit them. In this project we have built a scanner which provides malicious input to a website and lets the user know if the website has a vulnerability.

Contribution: Our tool tries to reduce the number of vulnerabilities in a web application to some extent by generating a report and lets the developer know what potential issues could be caused and remediate them before these vulnerabilities can be exploited. This tool could be used to detect and eliminate certain cases of injections but not all of them.

IV. Lab Setup and Running the Scanner

Resources:

1. Python installed in a Windows or Linux Machine with minimum 8 GB Memory and 256 GB Storage.
2. Python IDE to execute the program and run the scanner.
3. A Hypervisor to host a Virtual Machine [6].
4. The Metasploitable machine to a vulnerable website (DVWA) [7].

Setup:

1. Install the virtual machine application on your windows or Linux machine
2. Download a Python IDE
3. Download the Metasploitable machine
4. Before starting the Metasploitable machine using VirtualBox, go to settings and change the network settings to NAT network.
5. Once the network settings are changed, start the Metasploitable machine using virtual machine
6. Once the machine is running login into the machine using the following credentials-

   Username - msfadmin

   Password - msfadmin
7. After logging into the machine type "ifconfig" and press enter to check the IP address of machine

8. Now type the machine IP address in the host machine web browser to check if the host machine can communicate with the Metasploitable machine.

Running the scanner:

1. Before running the scanner, please run the following commands in the command prompt as the following two libraries are essential for the program to run.
   - pip install bs4
   - pip install requests
2. Once both the libraries are installed run the vulnerability_scanner.py program by executing the following command "python3 vulnerability_scanner.py".
3. The program should present the following options to check the vulnerabilities on
   - MyUCF
   - VulnWeb
   - DVWA
   - Custom URL
4. If the first option is selected the scanner will automatically scan for vulnerabilities in the *Vulnweb* application.
5. If the second option is selected the IP address of the DVWA machine needs to be provided as an input. This would scan for vulnerabilities in DVWA.
6. If the third option is selected the program would ask for the target URL, login, and logout URLs of the web application along with the username and password.
7. The output would print a list of vulnerabilities based on the option selected above showing different kinds of XSS and SQLi attacks that were successfully identified.

V. FEATURE ENGINEERING

When a user makes a certain request to a web server, the server responds with an output or asks the user for additional input before processing the request and returning a response to the user. While providing additional input, a user with malicious intent can inject certain scripts/payload, which if executed by the web server can lead to exploitation of a vulnerability. This could be due to not having essential input validation, conducting adequate testing and many such factors.

Generally, data is sent to a web application server in one of two ways

i. Through the URL
ii. Through the input elements/Forms.

The attacker can find a vulnerability via either of these two methods and exploit it either way, depending on the vulnerability.

The following are the steps that our tool follows to find and report vulnerabilities.

1) Crawling through the URL provided by the user

a) The program requires a target website i.e., the website URL on which the scanning needs to be performed. [2]

b) As the tool follows grey-box testing approach, along with providing the target URL, user login credentials are also required.

c) Login credentials of the user are required as the scanner can scan through all the URLs which a user can explore post signing in.

d) Once the tool has the required information, it starts gathering all the links that are associated with the provided target URL.

e) These associated URLs are stored for a later operation where malicious inputs / payloads are delivered to the website.

```
def crawl_URL(self, url=None):
    if url is None:
        url = self.target_url

    for link in self.extract_links(url):
        link = urlparse.urljoin(url, link)
        if "#" in link:
            link = link.split("#")[0]

        target_urls=self.target_url in link
        not_target_links=link not in self.target_links
        not_ignore_links=link not in self.links_to_ignore
        if target_urls and not_target_links and not_ignore_links:
            self.target_links.append(link)
            print(link)
            self.crawl_URL(link)
```

2) Extracting Forms from the webpages

a) Once the crawler collects all the URLs associated with the target website, the tool visits every URL and checks if there are any URLs and forms where a payload can be delivered. [3]

b) In order to find forms, the tool utilizes the *Requests* and the *BeautifulSoup* libraries.

c) The "*Requests*" library returns the entire HTML code of a webpage.

d) Once the HTML code of a webpage is received, the *Beautiful Soup* library filters out any forms that are present in the code.

e) These forms are later used to inject a payload and check if an attack was successful or not.

```
def extract_forms(self, url):
    response = self.session.get(url)
    html_code = response.content
    parsed_html = BeautifulSoup(html_code, "html.parser")
    return parsed_html.findAll("form")
```

3) ***Sending Payloads to discover vulnerabilities***

In this step we iterate over each URL and extract forms present in the respective URLs and deliver a payload into the URL or a form and to test. If any vulnerability is exploited by a payload, the tool returns the details of the webpage in which the vulnerability is present in along with the payload that exploited the vulnerability.

```
def submit_xss_form(self, form, value, url):
    method = form.get("method")
    inputs_list = form.findAll("input")
    post_data = {}
    for inputs in inputs_list:
        input_name = inputs.get("name")
        input_type = inputs.get("type")
        input_value = inputs.get("value")
        if input_type == "text":
            input_value = value
        post_data[input_name] = input_value
    if method == "post":
        return self.session.post(urlparse.urljoin(url, form.get("action")), data=post_data)
    return self.session.get(urlparse.urljoin(url, form.get("action")), params=post_data)

def submit_sql_form(self, form, value, url):
    method = form.get("method")
    inputs_list = form.findAll("input")
    post_data = {}
    for inputs in inputs_list:
        input_name = inputs.get("name")
        input_type = inputs.get("type")
        input_value = inputs.get("value")
        if input_type == "text":
            input_value = value
        post_data[input_name] = input_value
    if method == "post":
        return self.session.post(urlparse.urljoin(url, form.get("action")), data=post_data)
    return self.session.get(urlparse.urljoin(url, form.get("action")), params=post_data)
```

a) Discovering XSS and SQLi attacks in Forms

- After the tool performs Step - 1 and Step - 2, all the forms present in the associated URLs are gathered.
- Now the tool injects a list of payloads into the forms and submits the forms to the web server.
- Once the form is submitted, the tool checks for the submitted payload in response provided by the web server.
- If the payload is present in the response, then it means that the vulnerability was successfully executed.

b) Discovering XSS and SQLi attacks in URLs

- After performing Step - 1, the tool checks if there is any presence of "=" in the URLs.
- If the tool finds "=" in the URL, it will append a payload after the "=" and send the request to the server.
- As mentioned above, the tool then checks for the submitted payload in response provided by the web server.
- If the tool finds the payload in the received response, then a vulnerability was exploited.

```
for link in self.target_links:
    for form in self.extract_forms(link):
        for script in xss_scripts:
            print("--> Testing form in " + link)
            self.xss_tested.append(link)
            if self.test_xss_in_form(form, link, script):
                self.xss_found.append(link)
                print("\n\n***** XSS vulnerability discovered in "
                      + link + " in the following form *****")
                print("\n Script used --> " + script)
                print(form)
                print("\n\n")

            if "=" in link:
                print("\n\n--> Testing " + link)
                if self.test_xss_in_link(link, script):
                    print("***** Discovered XSS in " + link+ '*****')
                    print("\n Script used --> " + script)
                    print(form)
                    print("\n\n")
```

VI. EXPERIMENTAL RESULTS

The Table I. gives the information on the total number of known vulnerabilities found on different websites like DVWA [4] and VulnWeb [5] along with the total count, this information is used to compare the results and evaluate our tool.

TABLE I. Total number of known vulnerabilities in the following websites

Websites	XSS	SQLi	Total Vulnerabilities
DVWA [4]	2	3	5
VulnWeb [5]	10	17	27

The Table II. gives the information on the total number of vulnerabilities found by our tool on the vulnerable websites mentioned below

TABLE II. Total number of vulnerabilities found in the following websites

Websites	XSS	SQLi	Total Vulnerabilities
DVWA	2	4	6
VulnWeb	9	9	18

The accuracy of our tool in finding XSS vulnerabilities on DVWA is 100%. Although we were able to identify each website that contained a SQLi vulnerability, the tool found an additional web page on which there were no SQLi vulnerabilities according to the research article. Therefore, the accuracy of finding SQLi vulnerabilities on DVWA website is not 100% because it produces a false positive.

The accuracy of our tool in finding XSS vulnerabilities on Vulnweb is 90%, while the accuracy of finding SQLi vulnerabilities on VulnWeb is 52.9%.

While testing our tool, we happened to test on ***"my.ucf.edu"*** (For educational purposes only). Interestingly, we found an SQLi vulnerability in the course selection page of the website. Our tool was not further tested on any real websites due to the possibility that the payloads could cause repercussions to the websites or their backend databases.

VII. Future Work

A full working architecture is presented in the proposed system; however, some aspects can be modified for improved performance and accuracy of the scanning process. More vulnerabilities can be added to the scanning process to expand its scope even further. Coming versions will try to improve the accuracy of finding more SQLi vulnerabilities.

The scanner needs a more sophisticated crawling mechanism to guarantee that all the contents of a web application are scanned without leaving anything out.

With the addition of various payloads, it is possible for the scanner to find more vulnerabilities that can be exploited. The current version of the scanner does not work if there is any Multi-Factor Authentication enabled. Future versions of the scanner can handle these situations.

VIII. Conclusion

Despite the fact that this tool is useful in a variety of situations, it cannot always produce the same results on different websites. It is because they are not designed similarly and use different processes and logic. We believe that this is the reason why the accuracy of the vulnerabilities found on different websites is not consistent. As the tool is a generic scanner, it can be used to identify and remediate vulnerabilities to some extent. But for the tool to be extremely efficient it should be tailored according to a particular target website. This could mean changing the code of the tool where the tool can send requests to the website efficiently.

This paper shows how easy application-level vulnerabilities are to discover and exploit in large numbers of web applications by attackers. Attackers will soon start exploiting web vulnerabilities by using automated vulnerability scanning tools like VulnScanner, which we believe is only a matter of time.

We aim to raise awareness about the importance of integrating security into software development during the entire development process with this paper. Contrary to adding and testing it after the development cycle ends.

REFERENCES

[1] E. Gala´n, A. Alcaide, A. Orfila, J. Blasco. A Multi–agent Scanner to Detect Stored–XSS Vulnerabilities. University Carlos III of Madrid, UC3M Legane´s, Spain.

[2] https://pythonprogramminglanguage.com/get-links-from-webpage/

[3] https://www.thepythoncode.com/article/extracting-and-submitting-web-page-forms-in-python

[4] https://www.researchgate.net/figure/The-total-count-of-vulnerabilities-intentional-in-DVWA_tbl2_348147338

[5] https://logon-int.com/wp-content/uploads/2020/04/Developer_http_testphp_vulnweb_com_.pdf

[6] https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

[7] https://docs.rapid7.com/metasploit/metasploitable-2-exploitability-guide/