Web application vulnerability testing tool

Used site : testphp.vulnweb

**Project Objectives / Preview:**

The objective of this project is to assess the security of a web application by identifying common vulnerabilities that could lead to serious security incidents, such as data breaches, unauthorized access, and malicious attacks. The project also aims to:

* Automate the detection of various web application vulnerabilities.
* Provide actionable insights into improving the security of web applications.
* Help developers, security professionals, and organizations mitigate risks by highlighting critical security issues in their web applications.
* Ensure the application is protected against some of the most prevalent and damaging attack vectors.

This security scan is designed to be reusable across multiple web applications, making it a valuable tool for both small and large-scale web security assessments. The goal is to ensure that any web application subjected to this scan is thoroughly checked for common vulnerabilities, and detailed results are provided to facilitate patching and strengthening of security.

The code :-

The code file has been provided in the directory provided.

The explanation of the code :-

**Why Does This Matter?**

In today’s world, securing websites is crucial. Hackers are always looking for vulnerabilities to exploit, and if a site has holes like those checked by this script, it can lead to serious consequences—data breaches, loss of user trust, financial damage, and more. This script is a way to proactively find and fix those vulnerabilities before they are exploited.

**What This Code Does:**

Imagine you're a security expert or a developer, and you're trying to figure out if a website has any holes in its security. This code is like a tool that runs multiple tests on a website to check if it's vulnerable to certain types of cyberattacks. The script looks for common web vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), Server-Side Request Forgery (SSRF), and more. These are weaknesses that could be exploited by hackers to gain access to sensitive data or compromise the website’s integrity.

Let’s break down the script, step by step, so you can see exactly what each part is doing.

**Libraries We’re Using:**

1. **requests**:
   * **Why it’s here**: This is the go-to library for making HTTP requests in Python. It allows us to send requests to websites (like asking a site to give us its page content), which is how we interact with the website in the first place.
   * **How it’s used**: We use requests to send different types of HTTP requests (like GET or POST) to the target site. For example, when we’re testing if a website is vulnerable to SQL Injection, we send malicious payloads (e.g., SQL code) through these requests.
2. **BeautifulSoup from bs4**:
   * **Why it’s here**: BeautifulSoup is a fantastic tool for web scraping—this is when we pull content from websites and process it. In this case, we're using it to scan a website for HTML forms.
   * **How it’s used**: Once we load the page, BeautifulSoup helps us find all the forms on the page. Forms are key for testing things like Cross-Site Request Forgery (CSRF) vulnerabilities because attackers can trick users into submitting forms unknowingly. If we find any forms without security features, that’s a red flag.
3. **time**:
   * **Why it’s here**: The time library allows you to work with time, like adding delays between requests to avoid overwhelming the website.
   * **How it’s used**: Interestingly, it's imported but not used in this version of the script. It might have been intended for adding delays between requests, but in this case, the script doesn’t include any intentional pauses.

**The Vulnerability Tests:**

Now, let’s go through each of the tests the script runs. These tests look for common vulnerabilities that attackers might try to exploit.

**1. identify\_framework(url):**

* **Purpose**: Before we even start testing for vulnerabilities, it’s useful to know what kind of web framework the site is running. For instance, WordPress websites have their own set of common vulnerabilities.
* **How it works**: This function sends a HEAD request to the target URL to fetch only the headers (not the full content). It then checks if the server has a header like X-Powered-By that could tell us the technology stack. If the script doesn’t find that, it looks for WordPress-specific markers in the HTML itself.
* **Why it matters**: Knowing the framework helps attackers or security professionals tailor their attacks or tests. For example, if we know the site is running WordPress, we might check for common WordPress vulnerabilities.

**2. test\_sql\_injection(url, params):**

* **Purpose**: SQL Injection is one of the most dangerous vulnerabilities because it allows attackers to manipulate the database behind a website.
* **How it works**: The function tests if certain URL parameters (like id, user, or query) are vulnerable to SQL Injection by injecting common malicious payloads like ' OR 1=1 --. These payloads trick the website into executing unwanted SQL commands.
* **Why it matters**: If a site is vulnerable to SQLi, an attacker could steal data, delete records, or gain full access to the database. The script checks the server's response to see if any errors pop up, which would indicate that the site is vulnerable.

**3. test\_xss(url):**

* **Purpose**: Cross-Site Scripting (XSS) attacks involve injecting malicious JavaScript into a website. This can lead to all sorts of problems, including session hijacking and defacement.
* **How it works**: The function sends a variety of XSS payloads (e.g., <script>alert('XSS')</script> or <img src='x' onerror='alert(1)'>) through URL parameters to see if the site reflects these scripts back into the page. If the script executes, we know the site is vulnerable.
* **Why it matters**: XSS can be exploited to steal sensitive data like session cookies or to spread malware. So, identifying XSS vulnerabilities is crucial to securing the site.

**4. test\_ssrf(url):**

* **Purpose**: Server-Side Request Forgery (SSRF) lets an attacker make the server request resources from inside its own network, like internal APIs or services.
* **How it works**: The function sends a payload like http://127.0.0.1:22 (which refers to the local server) to see if the server responds in a way that indicates it's making internal requests.
* **Why it matters**: If an attacker can make the server connect to internal resources, they could bypass security controls and gain access to internal systems, making SSRF a dangerous vulnerability.

**5. test\_directory\_indexing(url):**

* **Purpose**: Directory Indexing vulnerabilities occur when a server accidentally lists the contents of a directory instead of showing an error page or redirecting to a proper page.
* **How it works**: The function checks if certain sensitive paths (like admin/, .idea/, or crossdomain.xml) are accessible and whether directory listings are enabled. If directory listings are allowed, an attacker could potentially view or download sensitive files.
* **Why it matters**: If directory indexing is enabled, attackers might find configuration files, backups, or other sensitive data that could lead to further attacks.

**6. test\_weak\_password(url):**

* **Purpose**: Weak passwords are an easy target for attackers. This function tries common username-password combinations to see if they work.
* **How it works**: It tests common usernames like test or admin along with common weak passwords like password or 123456. If any combination works, it means the site has weak authentication.
* **Why it matters**: Weak passwords are one of the easiest things to exploit. If attackers can guess the credentials, they can easily log into the site and steal data or make changes.

**7. find\_forms(url):**

* **Purpose**: This function checks for forms on the website, which can be targets for attacks like CSRF.
* **How it works**: It uses BeautifulSoup to parse the HTML of the page and find all the <form> elements. These are typically used for user input, and if they’re not properly secured, they can be exploited in CSRF attacks.
* **Why it matters**: If a form doesn’t have the right security measures in place, attackers could trick users into submitting malicious requests, leading to unauthorized actions.

**8. test\_csrf(forms):**

* **Purpose**: Cross-Site Request Forgery (CSRF) allows attackers to perform unauthorized actions on behalf of an authenticated user.
* **How it works**: This function checks whether the forms on the site include anti-CSRF tokens. These tokens ensure that the request is coming from a legitimate user and not an attacker.
* **Why it matters**: If forms are vulnerable to CSRF, attackers could trick users into performing actions they didn’t intend, like changing account settings or making purchases.

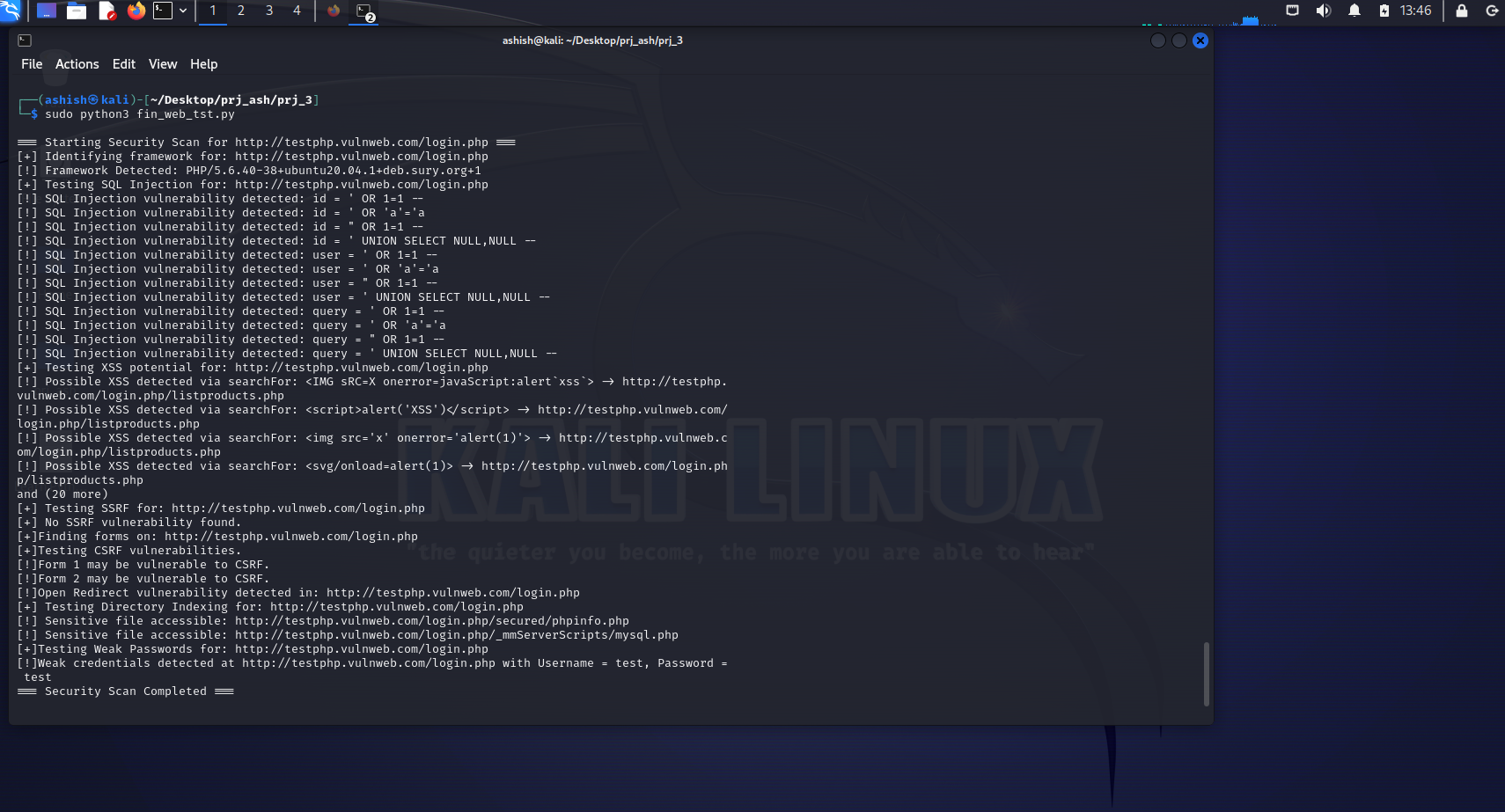
**9. test\_open\_redirect(url):**

* **Purpose**: Open redirects are a vulnerability where a site redirects users to another site, potentially a malicious one.
* **How it works**: The function appends a payload to the URL, like /?next=http://malicious.com, and checks if the website redirects to that malicious URL.
* **Why it matters**: Open redirects can be used in phishing attacks, where users think they’re visiting a legitimate site, but they’re actually being taken to a malicious one.

**Putting It All Together: run\_security\_scan(url)**

Finally, the script runs the run\_security\_scan function, which calls each of the individual tests in sequence. It prints out the results, showing any vulnerabilities that were found. The output gives the security professional or website owner an overview of what needs to be fixed to make the site more secure.

The output :-



The explanation of the output:-

**Starting Security Scan for** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

This line marks the beginning of the security scan for the web application located at http://testphp.vulnweb.com/login.php. The scan is looking for common vulnerabilities that could compromise the site’s security.

**Identifying framework for:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

The tool starts by identifying the underlying framework or technology the website is built with. This step is important because specific vulnerabilities are often tied to certain versions or types of frameworks.

**Framework Detected: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1**

The scan has identified that the web application is built using **PHP 5.6.40**. Knowing the version is crucial because older PHP versions are known to have vulnerabilities that could be exploited. For example, PHP 5.6.40 is an older version, and many security vulnerabilities are patched only in newer releases. This discovery could inform further security testing for known PHP-related vulnerabilities.

**Testing SQL Injection for:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

SQL Injection (SQLi) is one of the most common vulnerabilities. The tool is testing various parameters (id, user, and query) on the login page for potential SQL injection points.

**SQL Injection vulnerability detected: id = ' OR 1=1 --**

The first payload injected into the id parameter is a simple SQL injection attempt. ' OR 1=1 -- is a well-known attack that tricks the SQL query into returning data by always evaluating 1=1 as true. This means an attacker could bypass authentication or retrieve unauthorized information.

**SQL Injection vulnerability detected: id = ' OR 'a'='a**

This payload exploits the id parameter similarly, using a condition ('a'='a) that is always true, allowing an attacker to manipulate the SQL query.

**SQL Injection vulnerability detected: id = " OR 1=1 --**

Here’s another variation of the SQL injection, but this time using double quotes (") instead of single quotes ('). SQL injections can often be triggered by different types of quotes, and the fact that this also worked indicates that the web application does not sanitize inputs properly.

**SQL Injection vulnerability detected: id = ' UNION SELECT NULL,NULL --**

The UNION SELECT payload attempts to combine results from different queries. In this case, it selects NULL values, which would reveal the presence of a SQL injection vulnerability if it returns any output. This can be used to extract sensitive data from the database.

**SQL Injection vulnerabilities detected in user and query parameters**

The same set of payloads were tested for the user and query parameters, and vulnerabilities were found there as well. This means that the application is exposed to SQL Injection on multiple input fields, which is a major risk as attackers can easily manipulate these parameters to extract sensitive data or even compromise the entire database.

**Testing XSS potential for:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

Cross-Site Scripting (XSS) is a vulnerability where malicious scripts are injected into the web pages viewed by other users. These scripts can execute actions on behalf of the user, steal session tokens, or hijack user accounts.

**Possible XSS detected via searchFor: <IMG sRC=X onerror=javaScript:alertxss> ->** [**http://testphp.vulnweb.com/login.php/listproducts.php**](http://testphp.vulnweb.com/login.php/listproducts.php)

This output tells us that the searchFor parameter is vulnerable to XSS. The injected payload is an image tag that triggers a JavaScript alert when an error occurs. If this script executes on a victim's browser, it would indicate an XSS vulnerability. This attack could be used to execute arbitrary code in the victim's browser.

**Possible XSS detected via searchFor: alert('XSS') ->** [**http://testphp.vulnweb.com/login.php/listproducts.php**](http://testphp.vulnweb.com/login.php/listproducts.php)

Here, a simple <script>alert('XSS')</script> payload was injected. If the page renders this as executable JavaScript, it confirms that the site is vulnerable to XSS, and the attacker could execute arbitrary scripts.

**Possible XSS detected via searchFor: ->** [**http://testphp.vulnweb.com/login.php/listproducts.php**](http://testphp.vulnweb.com/login.php/listproducts.php)

This is another common XSS payload using an image tag. When the image fails to load (src='x'), the onerror event fires, executing the alert. If this triggers in the user's browser, it means the parameter is not sanitized, leaving the site vulnerable.

**Possible XSS detected via searchFor: <svg/onload=alert(1)> ->** [**http://testphp.vulnweb.com/login.php/listproducts.php**](http://testphp.vulnweb.com/login.php/listproducts.php)

Here, an SVG tag is used with the onload event to trigger an alert. SVG-based XSS is a more sophisticated attack, leveraging browser support for vector graphics, and it shows the variety of ways in which XSS can be executed.

**and (20 more)**

This line indicates that there were 20 more XSS vulnerabilities found with other payloads or parameters. The scan doesn’t show all of them for brevity, but it highlights that this application is highly susceptible to XSS attacks.

**Testing SSRF for:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

Server-Side Request Forgery (SSRF) is a vulnerability where an attacker can manipulate the server to make HTTP requests, often leading to internal resources being accessed.

**No SSRF vulnerability found.**

The scan didn’t find any SSRF vulnerabilities on this particular page, which is good because SSRF can be used to access internal services like databases or even trigger DoS attacks.

**Finding forms on:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

The script is looking for forms on the page. Forms are often the targets for exploits like Cross-Site Request Forgery (CSRF).

**Testing CSRF vulnerabilities.**

Cross-Site Request Forgery (CSRF) tricks authenticated users into performing unintended actions by submitting unauthorized requests. The tool checks if forms are protected against CSRF by verifying the presence of CSRF tokens.

**Form 1 may be vulnerable to CSRF.**

**Form 2 may be vulnerable to CSRF.**

The script found that two forms on the page may be vulnerable to CSRF, meaning they lack proper protection like CSRF tokens. This could allow attackers to impersonate users and perform unauthorized actions on their behalf.

**Open Redirect vulnerability detected in:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

Open Redirect vulnerabilities occur when an application improperly redirects users to an external site, often allowing attackers to send users to malicious websites.

**Testing Directory Indexing for:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

Directory indexing vulnerabilities expose sensitive directories or files, potentially leaking private data like configuration files.

**Sensitive file accessible:** [**http://testphp.vulnweb.com/login.php/secured/phpinfo.php**](http://testphp.vulnweb.com/login.php/secured/phpinfo.php)

This indicates that a file (phpinfo.php) is accessible to the public. This file typically contains detailed information about the PHP environment and can be exploited by attackers.

**Sensitive file accessible:** [**http://testphp.vulnweb.com/login.php/\_mmServerScripts/mysql.php**](http://testphp.vulnweb.com/login.php/_mmServerScripts/mysql.php)

The script also found another sensitive file (mysql.php) that could provide an attacker with database access or configuration details.

**Testing Weak Passwords for:** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php)

The script tests common username-password combinations to check if the site is susceptible to brute-force attacks or weak credentials.

**Weak credentials detected at** [**http://testphp.vulnweb.com/login.php**](http://testphp.vulnweb.com/login.php) **with Username = test, Password = test**

This result shows that the application accepts a weak combination of test as both the username and password. This is a major security issue because attackers can easily guess these credentials.

**=== Security Scan Completed ===**

The scan has now completed. The results indicate several critical vulnerabilities, including SQL Injection, XSS, CSRF, Open Redirect, Directory Indexing, and Weak Passwords. These vulnerabilities need to be fixed urgently to improve the site's security posture.

In summary, the scan found multiple significant vulnerabilities across the application, including SQL Injection, XSS, and weak credentials. It’s important to address these issues quickly to prevent attackers from exploiting them and potentially compromising the web application and its users.

**Project Summary:**

This project focuses on conducting a comprehensive security scan of a web application to identify common vulnerabilities that could jeopardize its integrity, confidentiality, and availability. The scan leverages various techniques to test for multiple security issues, including:

* **Framework Identification**: Identifies the underlying framework or technologies powering the web application.
* **SQL Injection**: Tests for vulnerabilities that allow attackers to manipulate the database by injecting malicious SQL commands.
* **Cross-Site Scripting (XSS)**: Checks for vulnerabilities where malicious scripts can be injected into web pages and executed in a user’s browser.
* **Server-Side Request Forgery (SSRF)**: Tests for weaknesses that could allow an attacker to make unauthorized server-side requests.
* **Cross-Site Request Forgery (CSRF)**: Verifies whether forms are protected from attacks that trick authenticated users into submitting unauthorized requests.
* **Open Redirect**: Tests if the application improperly redirects users to potentially malicious websites.
* **Directory Indexing**: Checks if sensitive files or directories are unintentionally exposed to the public.
* **Weak Passwords**: Assesses whether the application allows easily guessable or weak passwords.

The project employs a series of automated tests, primarily using Python scripts and libraries like requests and BeautifulSoup, to probe the target application for these vulnerabilities. The results help in pinpointing areas where the application’s security posture can be improved.

Conclusion :-

The security scan conducted on http://testphp.vulnweb.com/login.php uncovered several critical vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), Open Redirects, Weak Passwords, and CSRF, all of which could potentially be exploited by attackers to gain unauthorized access or cause significant damage. These findings indicate that the web application is not adequately protected against some of the most common and severe types of attacks.

To address these issues, it is highly recommended that the web application undergo a series of security improvements, such as input validation, proper sanitization of user inputs, protection against weak passwords, securing sensitive files, and implementing proper CSRF protection mechanisms. The identified vulnerabilities, if left unaddressed, pose a significant risk to the security and integrity of the application, and remedial actions should be taken as soon as possible.

By identifying these vulnerabilities early in the development and deployment process, developers and security teams can work proactively to mitigate potential risks and enhance the overall security posture of the application. This project serves as an essential tool in the ongoing efforts to create secure, resilient web applications in an increasingly complex cybersecurity landscape.