Spectre scan/report for testphp.vulnweb.com

Potential Attack Vectors Analysis

High-Priority Targets

- **`POST /search.php?test=query`**: A `POST` request to a search function with parameters is a classic entry point for both **Reflected Cross-Site Scripting (XSS)** and **SQL Injection (SQLi)**. The server is processing user-supplied input to generate a response.
- **`GET /artists.php?artist={id}`**: The `artist` parameter appears to be a numeric ID used to retrieve data. This is a prime candidate for **SQL Injection** and **Insecure Direct Object Reference (IDOR)** by manipulating the ID.
- **`GET /product.php?pic={id}`**: The `pic` parameter likely references a file or a database entry. This is highly suspicious for **Local File Inclusion (LFI)** if it's a file path, or **SQL Injection** if it's a database ID.
- **`POST /guestbook.php`**: A guestbook is a common target for **Stored Cross-Site Scripting (XSS)**, where a malicious payload is saved on the server and executed by any user visiting the page.
- **`POST /secured/newuser.php`**: An endpoint for creating a user located in a directory named `secured` is a major red flag for **Broken Access Control**. It must be tested to see if an unauthenticated or low-privileged user can access it.

Interesting Endpoints & Hypotheses

- **URL:** `http://testphp.vulnweb.com/search.php?test=query`
- **Hypothesis:** `Potential Reflected XSS or SQL Injection.`

- **Rationale:** A `POST` request to a search function (`search.php`) processes user input (`test` parameter). Input is likely reflected back on the results page (XSS) or used in a database query (SQLi) without proper sanitization.

- **URL:** `http://testphp.vulnweb.com/artists.php?artist=1`

- **Hypothesis:** `Potential SQL Injection.`

- **Rationale:** The `artist` parameter takes a numeric value (`1`, `2`, `3`). This pattern is common in applications that use the parameter directly in an SQL query like `SELECT * FROM artists WHERE id=1`. It's highly susceptible to injection.

- **URL:** `http://testphp.vulnweb.com/listproducts.php?cat=1`

- **Hypothesis:** `Potential SQL Injection.`

- **Rationale:** Similar to the `artists.php` endpoint, the `cat` parameter is used for filtering and likely fed directly into an SQL `WHERE` clause, making it a target for injection attacks.

- **URL:** `http://testphp.vulnweb.com/guestbook.php`

- **Hypothesis:** `Potential Stored Cross-Site Scripting (XSS).`

- **Rationale:** The endpoint is a guestbook, which implies user-submitted data is stored and displayed to other users. The `POST` request to this URL almost certainly corresponds to a form submission that could be poisoned with a malicious script.

- **URL:** `http://testphp.vulnweb.com/product.php?pic=1`

- **Hypothesis:** `Potential Local File Inclusion (LFI).`

- **Rationale:** The parameter name `pic` suggests it might be used to include an image file path on the page (e.g., `include('images/1.jpg');`). If not properly sanitized, this could be exploited to include and display sensitive server files.

- **URL:** `http://testphp.vulnweb.com/secured/newuser.php`

- **Hypothesis:** `Potential Broken Access Control.`

- **Rationale:** The URL contains the keyword `secured` and the title is `add new user`. A critical function like user creation should be restricted to administrators. This endpoint must be checked for unauthorized access.

- **URL:** `http://testphp.vulnweb.com/login.php`

- **Hypothesis:** `Potential for Brute-Force, SQL Injection, and Username Enumeration.`
- **Rationale:** As a standard login page, it is a critical entry point. The form fields (username, password) could be vulnerable to SQLi, and the server's response to failed logins could allow for username enumeration or brute-force attacks.

Exploitation and PoC

A01:2021 - Broken Access Control on `/secured/newuser.php`

1. **Objective:** Access the user creation page without authentication.
2. **Step 1:** Open a new incognito/private browser window to ensure you have no active session cookies.
3. **Step 2:** Navigate directly to the URL: `http://testphp.vulnweb.com/secured/newuser.php`.
4. **Verification:** If a form to "add new user" is displayed instead of a login prompt or a `403 Forbidden` error, the endpoint is vulnerable.
5. **Exploitation:** Fill out the form to create a new user (potentially with elevated privileges if possible) and submit it. If the user is created, the vulnerability is confirmed.

A03:2021 - Injection (SQLi) on `/artists.php`

1. **Objective:** Manipulate the SQL query to confirm the injection point.
2. **Step 1:** Navigate to a valid URL: `http://testphp.vulnweb.com/artists.php?artist=1`.
3. **Step 2 (Error-Based):** Inject a single quote `'` to break the query string.
    - **Payload:** `http://testphp.vulnweb.com/artists.php?artist=1'`
    - **Verification:** Look for a verbose database error message on the page (e.g., "SQL syntax error").

4. **Step 3 (Boolean-Based):** Confirm control over the query logic.
    - **Payload 1 (True):** `http://testphp.vulnweb.com/artists.php?artist=1 AND 1=1` (Page should load normally).
    - **Payload 2 (False):** `http://testphp.vulnweb.com/artists.php?artist=1 AND 1=2` (Page should load differently, e.g., no artist found).
5. **Automated Exploitation:** Use `sqlmap` to dump the database. `bash sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs --batch`

A03:2021 - Injection (Stored XSS) on `/guestbook.php`

1. **Objective:** Inject a persistent script that executes for any visitor.
2. **Step 1:** Navigate to `http://testphp.vulnweb.com/guestbook.php`.
3. **Step 2:** Locate the input fields for leaving a comment (e.g., "Name" and "Message").
4. **Step 3:** Enter a simple XSS payload into one of the fields.
    - **Payload:** `<script>alert('Stored XSS by Spectre')</script>`
5. **Step 4:** Submit the form.
6. **Verification:** Reload the `guestbook.php` page. If an alert box with the message appears, the payload has been stored and executed, confirming the vulnerability.

A03:2021 - Injection (LFI) on `/product.php`

1. **Objective:** Include and view a sensitive system file.
2. **Step 1:** Navigate to `http://testphp.vulnweb.com/product.php?pic=1`.
3. **Step 2:** Use path traversal payloads in the `pic` parameter to attempt to access files outside the intended directory.
    - **Payload:** `http://testphp.vulnweb.com/product.php?pic=../../../../../../etc/passwd`
4. **Verification:** Examine the HTML source of the returned page. If the contents of the `/etc/passwd` file (e.g., `root:x:0:0...`) are present anywhere in the response, the LFI vulnerability is confirmed.

Guide for Further Reconnaissance

1. **Automated Vulnerability Scanning:**
    - **Tool:** `OWASP ZAP` or `Burp Suite Pro`.
    - **Action:** Configure the scanner to proxy your traffic while you manually browse the site. Then, run an active scan on all discovered endpoints. This will automatically test for a wide range of vulnerabilities beyond the initial hypotheses.
2. **Directory & File Enumeration:**
    - **Tool:** `gobuster` or `dirsearch`.
    - **Action:** Brute-force common directory and file names to uncover hidden or unlinked content like admin panels, configuration files, or backups.
    - **Command:** `bash gobuster dir -u http://testphp.vulnweb.com/ -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x php,txt,bak,old,zip`

3. **Parameter Fuzzing:**

    - **Tool:** `wfuzz` or Burp Intruder.
    - **Action:** Systematically test all identified parameters (`test`, `artist`, `cat`, `pic`, `pp`) with curated wordlists for other vulnerability classes like Command Injection, Server-Side Template Injection (SSTI), or other edge cases.
    - **Command (LFI Fuzzing):** `bash wfuzz -c -z file,/usr/share/seclists/Fuzzing/LFI/LFI-Jhaddix.txt "http://testphp.vulnweb.com/product.php?pic=FUZZ" --hc 404`

4. **Authentication Mechanism Analysis:**

    - **Tool:** `Burp Suite Repeater`.
    - **Action:** Intercept the `login.php` POST request. Analyze the login logic for username enumeration by observing differences in responses for valid vs. invalid usernames. Test for weak password lockout policies by attempting a brute-force attack with a small wordlist.