AUTOMATED TEST REPORT

Table of Contents

Test Report

This report is the result of an automated scan performed with OWASP ZAP on http://127.0.0.1/, with the purpose of finding out whether the web application was vulnerable. The scan was performed with the help of OWASP ZAP 2.6.0. The report is formatted according to the NIST 800-115 guideline.

Executive Summary

The purpose of this automated test was to confirm whether the target web application meets the most basic security requirements. The test was meant to simulate a malicious attacker exploiting common vulnerabilities and breaches in a web application. The attacks were conducted under the level of access and privilege that a general Internet user would have.

Summary of Results

The automated scan found a total of 23 distinct vulnerabilities. The following types of vulnerabilities were found:

Cookie No HttpOnly Flag

A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
This vulnerability was found in the following URL(s):

http://127.0.0.1/

Web Browser XSS Protection Not Enabled

Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
This vulnerability was found in the following URL(s):

http://127.0.0.1/sitemap.xml

http://127.0.0.1/passwords/

http://127.0.0.1/config.inc

http://127.0.0.1/

http://127.0.0.1/javascript/

http://127.0.0.1/classes/

Application Error Disclosure

This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
This vulnerability was found in the following URL(s):

http://127.0.0.1/passwords/

http://127.0.0.1/javascript/

http://127.0.0.1/classes/

X-Content-Type-Options Header Missing

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
This vulnerability was found in the following URL(s):

http://127.0.0.1/robots.txt

http://127.0.0.1/passwords/

http://127.0.0.1/javascript/

http://127.0.0.1/classes/

http://127.0.0.1/

X-Frame-Options Header Not Set

X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
This vulnerability was found in the following URL(s):

http://127.0.0.1/passwords/

http://127.0.0.1/javascript/

http://127.0.0.1/classes/

http://127.0.0.1/

Directory Browsing

It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files , backup source files etc which can be accessed to read sensitive information.
This vulnerability was found in the following URL(s):

http://127.0.0.1/passwords/

http://127.0.0.1/javascript/

http://127.0.0.1/classes/

Attack Narrative

The scan was automatically executed with the help of OWASP ZAP 2.6.0. The scan consisted of a first stage of directory browsing (spidering) for a maximum number of 5 children directories. The second stage was an active scan that attempted to execute the following attacks in the target web application:

Attack	Description
Buffer Overflow	In the case of a buffer overflow, the extra data might get written into a sensitive memory location. To check for this flaw, the program sends large strings of input text and checks for anomalous behavior.
Client Browser Cache	Checks the headers of secure pages to ensure they don't allow browsers to cache the page.
CRLF Injection	CRLF refers to the Carriage Return and Line Feed special characters used to denote the end of a line. It might be abused by an attacker to split a HTTP response, which can lead to cross-site scripting or cache poisoning . The program tests for this vulnerability by trying to inject a new header into the response by sending a parameter preceded by CRLF characters.
Cross Site Scripting (Reflected)	First, it submits a 'safe' value and checks the response for any instances of this value. Then, for any locations where this value was found in, the program attempts a series of attacks aimed at that location to confirm whether a XSS attack is possible.
Cross Site Scripting (Persistent)	It begins by submitting a 'safe' value much like the reflected variant, but it then spiders the whole application to find any locations where the value might be found and tries to attack them.
Directory Browsing	Tries to gain access to a directory listing by examining the response for patterns commonly used with web server software such as Apache or IIS.
External Redirect	Tries to force a redirect by sending specific parameter values to the URL, and then checks the headers and response to check whether a redirect occurred.
Format String Error	Sends strings of input text that compiled C code would interpret as formatting parameters, and then looks for any anomalies.
Parameter Tempering	Submits requests with parameter values known to cause errors, and compares the response against patterns found in common errors in Java, Microsoft VBScript, PHP and others.
Path Traversal	Tries to access files and directories outside of the web document root using combinations of pathname prefixes and names of local files typically found in Windows or *NIX systems.
Remote File Inclusion	Passes an external URL as a parameter value for the current URL and checks whether this causes the title of the page to change.
Remote OS Command Injection	Submits *NIX and Windows OS commands as URL parameter values that would produce a specific output should the application be vulnerable. If this output isn't found in the response, then it tries a more complex approach that involves blind injection by submitting sleep instructions and timing the responses.
Server Side Code Injection	Submits PHP and ASP attack strings as URL parameter values and then examines the response.
Server Side Include	Detects the OS the server is running on, and then sends a HTML SSI directive. The response is then analyzed to check whether it matches with a pattern indicating the SSI (Server Side Include) was successful.
SQL Injection	Attacks URL parameters and form parameters with valid and invalid SQL syntax, using diverse SQL injection techniques such as error, Boolean or Union based injection.

Risk Rating

Vulnerabilities are rated according to their prevalence and their potential impact, as well as the difficulty of the exploit. The total number of vulnerabilities found for each risk level is:

- Low risk: 13
- Medium risk: 10
- Hight risk: 0

Recommendations

Given the vulnerabilities found, some possible recommendations to remediate each of them include:

Cookie No HttpOnly Flag

Ensure that the HttpOnly flag is set for all cookies.

Web Browser XSS Protection Not Enabled

Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.

Application Error Disclosure

Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

X-Content-Type-Options Header Missing

Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

X-Frame-Options Header Not Set

Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).

Directory Browsing

Disable directory browsing. If this is required, make sure the listed files does not induce risks.

Appendix: Methodology

It should be noted that this test was performed in an automated manner using tools such as OWASP ZAP, with no direct human intervention. This makes the test non-exhaustive, and it should be assumed that the target web application has many more vulnerabilities and security breaches than the ones mentioned in this report. This test should be taken as a 'lower floor' for possible vulnerabilities. If this automated test is able to find vulnerabilities in a web application, a resourceful human should be able to find many more, most of which would escape the scope of this test. If the tested web application is found to be vulnerable, it is recommended to contact a professional penetration tester to find and mitigate these and other security breaches.