Owasp PDF for Penetration testing
Recon and Discovery
(Specifically for web apps)
-Technology Stack
When testing a web application, you can use a tool called Wappalyzer to passively analyze the website and detect the technologies in use
You can also detect out of date JavaScript libraries using retire.js
-Service and vulnerabilities
A powerful scanner like nuclei is essential for discovering what might be running on an app including extracting version numbers, finding exposed admin panels, and scanning for known CVEs
This is how you can run nuclei to scan for known web based CVEs:
nuclei -target https://example.com -t http/CVEs
-Content discovery:
We can discover the content of a web application mainly using two popular methods. First, we'll take a look at spidering. Spidering basically follows links of a web page to find more links. Spidering tools include gospider
We can also discover content through directory brute forcing. Directories are essential components that organize content in web applications. For instance, each category that is organized in such a hierarchical structure has its own URL. Directory brute forcing involves repeatedly trying different combinations of words to uncover hidden files. For effective directory brute forcing, we need good wordlists. You can grab wordlists from http://wordlists.assetnote.io/ as well as https://github.com/danielmiessler/SecLists/tree/master/Discovery/DNS . You can also make your own wordlists based on your needs and the target you intend to hack. Examples of tools to perform brute forcing include feroxbuster and dirbuster among many others.
Static analysis:
This type of testing encompasses thoroughly analyzing source code in order to find vulnerabilities. You can find functions that may lead to the exploitation of a cross site scripting attack. Semgrep is a tool that uses templates to uncover vulnerabilities in source code. Semgrep can be easily installed as a Python3 pip module:
pip3 install semgrep
We also have another tool known as trufflehog that can detect secrets in source code, S3 buckets, and git repositories.
-Fingerprint web server:
It's finding out the type and version of web server a target is running. Determining the version and the type of a running web server can enable us to further discovery of any know vulnerabilities. In particular, servers running older versions of software can be susceptible to known version specific exploits.
How to test:
-Banner grabbing which is basically examining the response header of an HTTP request. Sometimes the version is exposed. However, some applications obfuscate their server information by modifying the header.
We can also send malformed requests and examine their error responses as error pages tend to differ between different types of web servers.

We can use automated tools to produce more accurate results of web server finger printing
For example, Nmap and netcraft
Review webserver metafiles for information leakage:
We can test information leakage by testing the robots.txt file through multiple ways
Robots.txt
Web spiders crawl web pages by traversing through hyperlinks. Their accepted behaviour is specified by the robots.txt file. For instance, the robots file at the root of the web directory can specify that the web crawler must not access certain resources using the disallow directive followed by /resource.name. However, robots can intentionally ignore these directives.
How to retrieve the robots.txt file:
wget http://google.com/robots.txt
Or
curl -o http://google.com/robots.txt
We can use rock spider to create the initial scope for the web crawler.
You can use the Google Webmaster Tool to analyze a website
Enumerate applications on web server
A paramount step to do while testing for vulnerabilities is to find out which applications are running on a web server.
There are three factors influencing how many applications are related to a given domain name (or an IP address)
1)Different base URL
For instance, www.example.com may be associated with http://www.example.com/ur1, http://www.example.com/ur2, and http://www.example.com/ur3.
In this case, these pages aren't hidden but their location isn't explicitly advertised.
2) non-standard ports
While web applications usually run on ports 80 and 443, some web apps are associated with arbitrary TCP ports and can be referenced by specifying the port number as follows:
https://www.example.com:port/
3) Virtual hosts
A single IP address can be associated with 9one or more symbolic names. For instance, a certain ip address such as 145.175.199.100 may be reserved for example.com, helpdesk.com, and webmail.com
One would not suspect the existence of other web applications in addition to example.com.

How to address issue 1
If testers suspect the existence of hidden web applications, they could search using the site operator and examine the result of a query for site: example.com
We can do a bit of dictionary guessing to yield some results by probing for URLs that may point to non-published applications. For example, a web mail front end may be accessible through https://example.com/webmail or https://mail.example.com . This is also true for administrative interfaces.
How to address issue 2
We can check for web applications running on non standard ports by using the -sV option in nmap.

How to address issue 3
There are many ways such as DNS zone transfers and DNS inverse queries. You can also use Reverse IP services such as Webhosting info that reveal non obvious symbolic names mapping the one IP address.

Review web page comments and metadata for information leakage
Comments and metadata included in HTML code might reveal information that shouldn't be available to attackers.
Html comments are sometimes used by developers to include debugging information about the web application. It may also be SQL code, usernames and passwords, or internal IP addresses. Check for the comment tag <!-- ..--> in the HTML source code to find comments. Further, a meta tag may include information related to the application. For instance, meta tags can be used to specify keywords that a search engine may use to improve the quality of search results.
Identifying application entry points
Before we undertake testing of the web application, we need to enumerate it and its attack surface. To do so, we must understand how requests are formed and typical responses from our target application. First, we need to pay attention to all the HTTP requests as well as every parameter and form field passed to the application. We must also pay special attention to GET requests, POST requests that are used to pass arguments to the application, and when other methods for RESTful services are used.
In order to see the parameters inside the body of a request, we need to use a tool known as an intercepting proxy. Within the post request, the tester must take special note of hidden form field that are passed to the application and may contain sensitive information such as quantity of items or price of items.
For requests:
Identify when GET and POST request are used
Identify hidden parameters in POST requests
Identify all parameters used in GET requests particularly the query string (usually after a ?)
Identify all parameters of the query string. Also note that many parameters can be used in one query string separated by a &, \- , :, or any other special characters.
For responses:
Identify where new cookies are set (Set-cookie header), modified, or added to.
Identify where there are 400 status codes, in particular 403 forbidden, and 500 internal server errors(during normal responses)
OWASP attack surface detector
A tool that investigates the source code and uncovers the endpoints of a web application, the parameters these endpoints accept, and the date type of those parameters. It is available as a plug-in for burp suite or as a CLI tool.
Map Execution paths through application
In order to thoroughly test an application, we first need to understand its structure. For this, we must document the discovered code paths. A good way to do this is to start with a spreadsheet and document all the links discovered by spidering the web application and then scrutinizing these links and adding details and screenshot descriptions of the paths discovered. There were a wide range of spidering tools that we can use including ZAP (zed attack proxy)

Fingerprint web application framework

A web application framework is a piece of software that facilitates the development of web applications by providing a range of tools and tasks involved in the web app development process. Fingerprinting the web application framework is important if the tester is already familiar with the framework of the target application (known vulnerabilities in unpatched versions, specific misconfigurations, and known file structure)

How to define the current framework:

HTTP headers

Look at the X-Powered-By field in the HTTP response header. However, through a simple configuration, the owners can obfuscate these headers. Sometimes, the X-Generator header may point out to the used framework.

Cookies

Another more reliable way to determine the current web framework is to find framework specific cookies.

Framework	Cookie
Zope	zope3
CakePHP.	cakephp
Kohana.	kohanasession
Laravel.	laravel_session

HTML source code

This technique is based on finding certain patterns in the HTML source code. One of the common markers are HTML comments that disclose the used framework. More often we can also find links to framework specific CSS or js folders. Also, certain script variables may point to a certain framework. More frequently such information is placed between the head or meta tags or at the bottom of the page.

General markers:

%framework name%

Powered by

Built upon

running

Error messages

Specific files and folders

Which are different for each specific framework.

File extensions

The URL may include file extensions which can also help identify the web platform or technology.

Here are some common web extensions and technology:

php - PHP

aspx -Microsoft ASP.NET

jsp - Java server pages

Tools:

What web (included by default in Kali Linux)

Wappalyzer (Firefox Chrome extension)

Fingerprint Web Application
How to identify the web application (or the CMS used) to catch vulnerabilities
Cookies
Check for application specific cookies
HTML source code
Check the whole document to find useful information
Specific files and folders
Every application has its own specific file and folder structure on the server. In order to uncover them, a technique known as dirbusting is used. Dirbusting is brute forcing a target with predictable folder and file names and monitoring the HTTP responses to enumerate serover contents. This information can be used for both finding default files and attacking them, and for fingerprinting the web application.
Several good file lists already exist and one good example is FuzzDB wordlists of predictable files/folders.
Map application architecture
The application architecture needs to be mapped through some tests to determine the different components that are used to build the web application. In order to extend the architecture map, we can ask ourselves multiple questions. Is there a firewall protecting the web server? This can be answered by the result of port scanning.
We can also detect reverse proxies that act as IDS if the target returns a different error message for some common web attacks than the one usually given by the web server to a request that targets unavailable pages (404 message).
Configuration and deployment management testing
Test network infrastructure configuration
Configuration management and review is a fundamental step in application deployment. A single misconfiguration can undermine the security of the entire infrastructure and introduce undesirable implications.
In order to test the configuration management, we need to determine the different elements that make up the infrastructure and review them in order to make sure they don't contain any vulnerabilities. We also need to review the administrative tools, the authentication systems, and a list of defined ports which are required for the application.