

# Incorporating Web Application Scanning into a Vulnerability Management Program: An Operational Guide

## Positioning DAST within the Enterprise Security Toolkit

### Introduction

This document provides guiding principles for integrating a *Dynamic Application Security Testing* tool (DAST)<sup>1</sup> into routine *Enterprise Vulnerability Management*<sup>2</sup> processes. The scope and responsibilities of teams constituting the *Vulnerability Management* function are frequently subject to discussion and adjustment. Each organization must tailor its approach to implementation, tool selection (such as whether or not to use DAST), and resource allocation based on aligned and adjacent functions, budget constraints, and strategic priorities.

For the purposes of this document, *Web Application Scanning* will refer specifically to the comprehensive analysis of live HTTP application interfaces, as distinct from the static analysis of source code or configuration files (SCA<sup>3</sup>). While the terms “*Web Application Scanning*” and “*DAST*” may be used interchangeably herein, we primarily let DAST denote the toolset or capability, whereas *Web Application Scanning* refers to the operational practice.

This review examines the role of active *Web Application Scanning* within the broader context of vulnerability assessment, highlighting the introduction and understanding of features such as *web fuzzing*, *application crawling*, *DOM state manipulation*, and *automated web application authentication*.

Although some references to *Tenable*'s<sup>4</sup> solutions will appear throughout, the ambition is to let the discussion remain primarily vendor-neutral.

© 2025 Sven Berglund. All rights reserved.

If you wish to republish this material, please provide attribution and a referral link to the original article at <https://certerion.com>.

- 1 DAST stands for *Dynamic Application Security Testing*. It is a security testing methodology that analyzes applications (typically web applications) while they are running, from an external perspective to identify vulnerabilities. See e.g. <https://owasp.org/www-project-devsecops-guideline/latest/02b-Dynamic-Application-Security-Testing>
- 2 For discussion around the *Vulnerability Management* function, see Appendix E: *Expanding on Common functions and perspectives on Web Application Scanning*
- 3 SCA stands for *Software Composition Analysis*. It is an automated process that identifies all open source and third-party components within a codebase, typically analyzing the code and configuration at rest directly from its repository checking them for known security vulnerabilities and license compliance issues
- 4 <https://www.tenable.com/products>

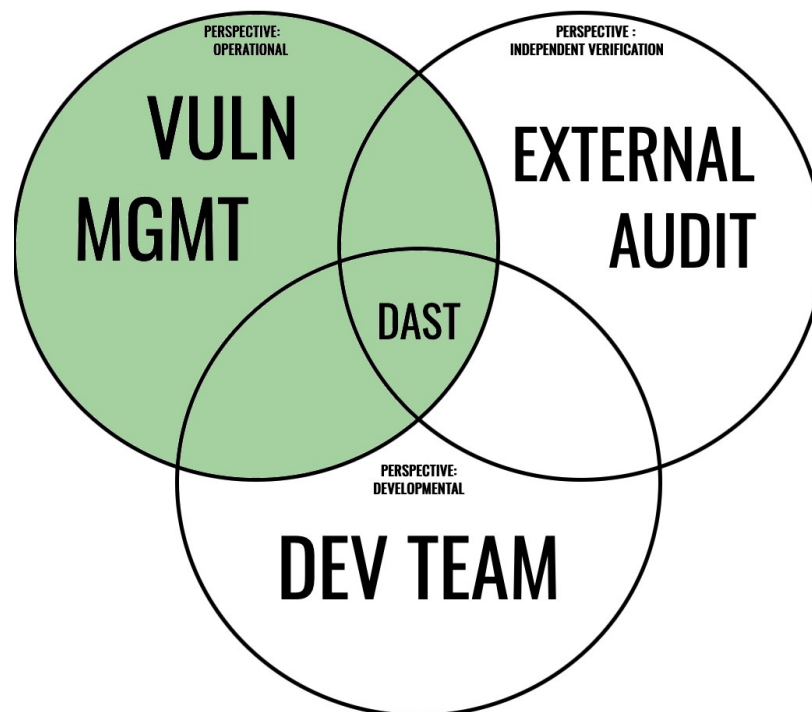
## Table of Contents

Incorporating Web Application Scanning into a Vulnerability Management Program: An Operational Guide.....	1
Introduction.....	1
Dynamic Web Application Scanning – The duty of whom?.....	3
Dynamic Web Application Scanning vs. General Network based Vulnerability Scanning.....	4
Common program considerations.....	6
Operational aspects to resolve.....	6
Application prioritization.....	7
Scan Depth and caution.....	10
Large environment application discovery.....	11
Appendices.....	12
A. Schema for classification: <i>Scan depth and caution</i> .....	12
B. Schema for classification: <i>Web Application Customization Spectrum</i> .....	15
C. Expanding on <i>Web App Scanning vs. Network Vulnerability scanning</i> .....	17
C.1. Targeting specific apps by URL.....	17
C.2. DOM interactions and DOM aware scanning.....	18
C.3. Active Probing techniques.....	19
C.4. Credentialed scanning with a DAST tool.....	21
Toolkit rather than ascertained feature support.....	22
C.5. Ideal Scan cycle implementation may differ.....	23
C.6. Specialized Focus: Vast framework and library recognition.....	24
D: Unpacking <i>Active vs. Passive probing</i> .....	25
E: Expanding on <i>Common functions and perspectives on Web Application Scanning</i> .....	27
End notes.....	28

## Dynamic Web Application Scanning – The duty of whom?

Web Application Scanning (DAST) is typically performed by some or all of the 3 functions:

- **Vulnerability Management**
- **Development team (or SDLC aligned QA function)**
- **External Auditor or pentester**



*Figure 1: Three perspectives on web application security testing, with the main functions owning these perspectives. Operational perspective (Vulnerability Management function), Developmental perspective (Dev/QA team function), and Independent verification perspective (External Audit function or Pen Testing function).*

The boundaries between the practices of these functions can overlap significantly and Web Application Scanning is one such area of overlap. Compliance requirements can become a key driver for implementing Web Application Testing in all of the 3 functions<sup>1</sup>.

In this document, as the title indicates, we focus on incorporation of Web Application Security Assessment of the DAST type into the Vulnerability Management function. We try to address typical questions that tend to arise in the implementation.

**Drill down:** 🔧 See Appendix: [E: Expanding on Common functions and perspectives on Web Application Scanning](#) for more discussion around the 3 different functions/perspectives depicted in the diagram.

# Dynamic Web Application Scanning vs. General Network based Vulnerability Scanning

Since running generalized or multi-purpose vulnerability scans of IT-Assets over the network (using tools such as Nessus<sup>5</sup>) often constitutes the backbone of a Vulnerability Management program, the introduction of a DAST<sup>1</sup> tool can sometimes cause confusion or be mistakenly deemed as a redundant assessment method.

## Examples of [broad-scope] *Enterprise Vulnerability Scanners*

*Nessus (Tenable), QualysGuard (Qualys), InsightVM (Rapid7), OpenVAS, Nexpose (Rapid7), Languard (GFI)*

## Examples of *DAST* tools/platforms

*Tenable Web App Scanning (Tenable), OWASP ZAP, Burp Suite (PortSwigger), Invicti DAST (Invicti), Acunetix (Invicti), Veracode Dynamic Analysis (Veracode), InsightAppSec (Rapid7)*

**Disclaimer:** When compiling this document *Tenable's* implementations have been used as points-of-references. The exact feature separation between DAST and more general-purpose scanning might not be applicable in exactly the same way in one-to-one comparisons between all of these tools.

The tools to conduct *DAST* versus *General-purpose Vulnerability Scanning* can sometimes be architecturally very similar<sup>6</sup> but they provide different capabilities. These are 6 common characteristics that highlight the contribution of a DAST tool in the Enterprise Security Toolkit:

### ➤ 1. It's Targeting specific apps by URL

Where the typical broad-spectrum enterprise vulnerability scanning focuses on entities like *IP*, the *Host* or the *Asset*, a DAST tool will target *URLs* representing individual *Applications*.

Drill down:  See *C.1. Targeting specific apps by URL*

### ➤ 2. It executes DOM aware scanning

Unlike the capabilities of most general-purpose vulnerability scanners the dedicated DAST tool must be able to expand, manipulate and crawl the DOM (Document Object Model<sup>7</sup>) presented by the target application. This is a necessity for in-depth scanning and comprehensive vulnerability enumeration for

<sup>5</sup> <https://www.tenable.com/products/nessus>

<sup>6</sup> There are architectural similarities e.g. in how one can link both a Nessus Scanner and a Web Application Scanner to Tenable Vulnerability Management, running on very similar appliances and with similar client-to-API polling architecture.

<sup>7</sup> <https://www.geeksforgeeks.org/html/dom-document-object-model/>

the Application and it's inner workings.

Drill down:  See **C.2. DOM interactions and DOM aware scanning**

➤ **3. It performs a more active and potentially offensive probing of applications**

Be aware of the DAST tool's potential offensiveness given it's various configuration options. Some categories of tests, such as inquisitive injection and XSS tests on custom code bases cannot be performed without attempting to actually inject data.

Drill down:  See **C.3. Active Probing techniques**

➤ **4. It's Credentialed scan support needs to be understood in the context of each application**

Credentialed scanning of a web application UI is a vastly different enterprise than credentialed scanning of a host, a database, or even an API. It is by nature an abusive practice and feasible methods to achieve successful credentialed scan coverage will vary depending on the target site architecture.

Drill down:  See **C.4. Credentialed scanning with a DAST tool**

➤ **5. It's scan cycle benefits from being coordinated with the SDLC (Software Development Life Cycle<sup>8</sup>) [if possible] rather than merely the enterprise patch management cycle**

In those cases when we employ DAST as a response to in-house development, we want to optimize the way to feed back the results to the Dev function, rather than merely reporting to the general patch management function.

Drill down:  See **C.5. Ideal Scan cycle implementation may differ**

➤ **6. As a product it has a more Specialized focus**

Investment in a DAST tool will build on top of your vulnerability management stack and contribute with a focus on web app security that other scanning tools will not match.

Drill down:  See **C.6. Specialized Focus: Vast framework and library recognition**

---

8 <https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/>

## Common program considerations

Some typical considerations tend to appear when strengthening the Web Application aspect of a Vulnerability Management program. Many of these become particularly obvious in large organizations, where the Vulnerability Management function must systematically discover and catalog thousands of assets of various types, many of those with web interfaces. And prioritize not only remediation efforts but also the actual assessment types (Scan types and cadences) and their scheduling across diverse web assets and other assets. In this process we need to distinguish the assets where Dynamic Web Application Scanning is actually motivated and essential.

Important questions to ask regarding internal processes and overarching business circumstances affecting the resulting operations, will include:

1. Do we build Applications ***in-house***?
  - In that case, how are development teams integrating DAST workflows? – As isolated DevOps processes *or* as federated within the Enterprise Vulnerability Management program?
  - Do we have access to test environments?
2. What *protocols, compliance requirements* and *SLAs* govern the assessment of ***externally developed*** web applications and APIs?
3. Is the Vulnerability management function tasked with ***prioritizing*** how a DAST tool license is utilized, i.e. what applications to scan? Or is the utilization scope of the DAST investment given to the Vulnerability Management team as a premise up front, with a list of target systems?
  - In the case of prioritization, does the vulnerability Management function have (and are able to maintain) a reliable catalog of exposed applications with web interfaces?
4. Do we have ***permission*** for running the scans we need to run?

For larger organizations needing to streamline processes and practices while managing vast numbers of digital assets these questions usually require a bit more attention. When the above questions have been explored, next we will look at 3 important **operational** aspects that we will need to resolve and some guidance on how to tackle them.

## Operational aspects to resolve

While the relevance of *Application Prioritization* and *Application Discovery* may depend on organizational context (usually being more pressing in larger organizations), *Scan Depth and caution* remains a more universal consideration for all vulnerability management programs using DAST.

- ***Application prioritization***
- ***Scan Depth and caution***
- ***Large environment application discovery***

## Application prioritization

In those situations where the Vulnerability Management function has a large inventory of web interfaces and is tasked with prioritizing which ones to target for web application Scanning (and many times to allocate a DAST tool license), here are some dimensions that can be relevant in such a prioritization process.

We will choose to look at the 4 dimensions:

- *Degree of customization*
- *Exposure*
- *Backend sensitivity*
- *Duty of Assessment*

Starting on the next page with the very important *Degree of customization* i.e. the answer to the question: “*To what extent does the app build on a custom code base?*” Some organizations will indeed use this criterion exclusively (limiting the scope to a set of self-developed apps only) others will see the need to weigh in other *dimensions* in the prioritization as well, e.g. the remaining 3 above.

It's not difficult to come up with yet more candidate dimensions to include, they could be e.g.

- *Hosting*: Part of a SaaS package – Contracted – In-house (on-prem or IaaS)
- *Liability*: What is the liability if compromised

## Degree of customization

The **degree of customization** is a very important facet if we are tasked with prioritizing usage of a DAST investment among a large set of web interfaces. We need to be able to estimate this degree accurately. To do this in a consistent way can be a challenge when we have to deal with a large number of detected web application interfaces (maybe in the hundreds) in larger corporate environment. Those application interfaces are often distributed on a wide range spanning from SaaS<sup>9</sup> interfaces to homegrown heavily customized web applications.

We can do the following classification, where the further down on the scale a web interface gets placed, the more imperative it is to have it scanned in-depth.

### Web application customization spectrum

	<b>Customization of:</b> →	<b>Deployment configuration / Hosting environment / Update management</b>	<b>Contributed modules</b>	<b>Application Logic</b>	<b>Web Framework internals</b>
1	Off-the-Shelf SaaS				
2	Configured Commercial Products	X			
3	Extended Platform Solutions	X	X		
4	Framework-Based Custom Applications	X	X	X	
5	Framework-Less Custom and Legacy Applications	X	X	X	X

The classes 5, 4 and 3 are usually the ones prioritized for being scanned with a DAST tool.

#### Drill down:

We elaborate further on this classification schema in Appendix [B. Schema for classification: Web Application Customization Spectrum](#) where we describe the classes from 1-5 and list a few typical examples of each.

<sup>9</sup> Software as a Service, see e.g. <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas>



## Exposure

Another common prioritization criterion is *Exposure*. How exposed is the web application interface, is it well protected in limited internal network environments only? Or is it exposed world wide on a public IP? To what extent is the access restricted with components and configurations such as Access portals, WAFs, Federated logins, IP Restrictions and ACLs.

<b>Limited Internal exposure</b>	Intranet, Extranet or wide site-to-site exposure	<b>World-wide external (public) exposure.</b>
----------------------------------	--	---



## Backend Sensitivity

To what extent is the Backend sensitive, as a point of lateral movement, a point of data corruption or leak, a point of malware deployment, etc.

<b>Lower extreme of backend sensitivity</b>	<b>Higher extreme of backend sensitivity</b>
Static web page with non sensitive information	Mission critical system or integration, highly business critical data, highly sensitive data.



## Duty of Assessment

To what extent is the Vulnerability Management function the accountable owner of the Web Application Scanning? Is it an overlapping duty with other functions (as described in the Venn diagram at section *Dynamic Web Application Scanning – The duty of whom?*).

<b>Other function(s) tasked with assessment</b>	<b>Exclusively Vuln Mgmt</b>
There are other assessment functions in the organization responsible for DAST, such as <a href="#">SDLC aligned teams</a> .	The Vulnerability Management is the solely accountable function within the organization, tasked with performing DAST scans on web applications.



## Scan Depth and caution

When tasked with scanning a set of web applications, for each of those we need to decide on the scan depth and caution.

Different dimensions of scan depth:

- Are we crawling<sup>10</sup> the app or scanning a fixed select number of internal paths?
- Are we running credentialed scans?
- Are we scanning full-blown with exhaustive assessments/plugins or using a more cautious approach?
- Are we scanning production environment, test only or both

As discussed in the section *Active Probing techniques: Offensiveness* the DAST tool will typically be configurable into a wide spectrum of different kinds of scans ranging from a very simple and quick request of the target site, to a potentially offensive crawling scan that can go on for hours.

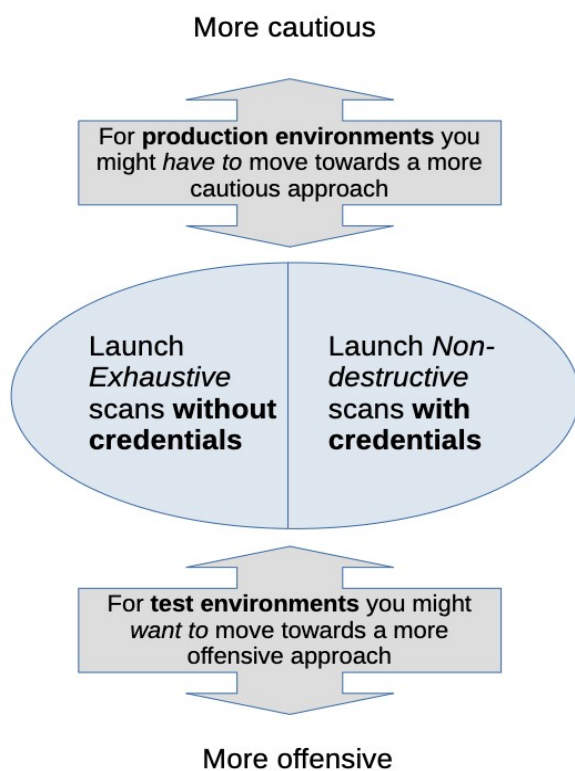


Figure 2: Depiction of a common middle-ground and default approach in web application scanning. Avoiding to scan offensively with credentials.

### Drill down:



What can be a systematic way to define our *Exhaustive* vs. *Non-destructive* Scans?

What would a more *Cautious* vs. a more *Offensive* approach look like?

In Appendix [A. Schema for classification: Scan depth and caution](#) we define a method for classifying scan depth and caution In 5 different levels, ranging from *Cautious* to *Offensive*.

The concept is to provide a structured set of definitions where a large organization can define it's general approach for both Test- and Production environments and then roll out this approach in a systematic way.

## Large environment application discovery

In some scenarios the license for Web Application Scanning is purchased for specific purpose. In others, the Vulnerability Management function may be tasked with listing and maintaining the inventory of web application interfaces in order to proceed with prioritizing which applications to scan. In large environments we can use tools such as:

**Network scanning tools** (Nessus Discovery Scans and similar) – Filter cumulative scan data e.g. with Common web application ports and Web application detection plugins to aggregate lists of web application interfaces.

**For external (public DNS and IP surface) you can utilize tools crawling public registries** of DNS and ASN (Anonymous System Numbers) as well as services utilizing web spiders to collect information on exposed web interfaces.

Using tools like this, we can often enumerate many hundreds of web application interfaces in larger organization's public IP inventories, expanded from a handful of subdomains. However, scanning all discovered web interfaces with a DAST is typically not practical. Hence, in those situations the next step will be to prioritize or triage the discovered web application interfaces e.g. according to the previous section *Application prioritization*.

### Tenable implementation

Implement organization-wide Discovery scanning

<https://docs.tenable.com/vulnerability-management/Content/Scans/DiscoveryAssessmentScans.htm>

External enumeration with ASM (Attack Surface Management) can be used additionally to map up the inventory from the organization's public footprint.

<https://www.tenable.com/products/attack-surface-management>




From both of these sources lists of web interfaces can be extracted

# Appendices

## A. Schema for classification: *Scan depth and caution*

In order to define a systematic approach for a Vulnerability Management function, *in terms of what Scan Depth and Caution to implement with your DAST tool*, we will define a classification of 5 different levels, ranging from *Cautious* to *Offensive*.

Let's start by defining 3 conceptual classes of web application scans:

	<b>“Quick and shallow”</b> – A non crawling <sup>10</sup> shallow scan that does not impact the target site any more than requesting the URL in a browser. This type of scan can do some basic assessment of TLS/SSL configuration, Certificates and Http Headers and will never have any negative impact on a production site.
	<b>“Non-Destructive”</b> – A more extensive scan, potentially crawling the site. Avoids assessments of type <i>injection tests, fuzzing and offensive XSS tests</i> . It can still be very revealing and typically run analyses of used frameworks, referenced libraries and dependencies. Such as assessments of the type discussed in section <a href="#">C.6</a> . <i>Specialized Focus: Vast framework and library recognition</i>
	<b>“Exhaustive”</b> – A scan with all features enabled, including such more offensive active tests that can impact the state of the application, create or corrupt persisted data. See discussion in sections: <a href="#">C.3. Active Probing techniques</a> , <a href="#">Active Probing techniques: Offensiveness</a>

⚠ Note that due to the risk of inadvertent DOS effects we might want to schedule even a *Non-destructive* scan nightly or at least during off-peak-hours - *if* launching it against production environments. For a *Quick and Shallow* scan, by definition we need not to worry about any effects on target sites.

### Tenable implementation

Predefined scan types:

<https://docs.tenable.com/web-app-scanning/Content/WAS/Scans/ScanTypes.htm>

Templates:

<https://docs.tenable.com/web-app-scanning/Content/WAS/Scans/ScannerTemplates.htm>

(The plain “Scan” template by default is an exhaustive scan with every plugin enabled, however it is

<sup>10</sup> We use the term “crawling” in the meaning of extracting, indexing and following internal http links when scanning a web application. See e.g. <https://www.techtarget.com/whatis/definition/crawler>

also the most configurable with all settings available)

<https://www.tenable.com/plugins/was/families>

**Remark:** If you need to craft a non-destructive scan with support for credentials, then you might have to start with the plain “Scan” template and disable a number of plugin families, see for example what has been left out from the “Basic” scan configuration among the predefined types.

See also resources like this:



<https://www.youtube.com/watch?v=vo89x18JrzE&t=275s>

The core principle is that the middle ground (*Labeled as the “Confident” approach on the next page*) usually is a good default level with a balance between caution and offensiveness. However, for test environments you might *want to* move down in the hierarchy towards a more *Offensive approach*. And for prod environments you might *have to* move upwards towards a *Cautious approach*.





## 5 approaches to scanning our web app environments with a DAST tool

### Cautious approach

Non-Credentialed surface (Optionally) <i>Quick and Shallow</i>	
Credentialed surface	
No scanning	



The cautious approach is to do **no scanning at all** on our production environments, or to allow only a very light and superficial assessment like **Quick and Shallow** as described above. For in-house developed apps where we have access to scanning a test environment more thoroughly, such a very limited scanning of a production environment need not be a big issue.

### Moderate approach

Non-Credentialed surface <i>Non-Destructive</i>	
Credentialed surface (Optionally) <i>Non-Destructive</i>	

In many scenarios, for example when we don't have access to test environments, we want to do more than Quick and Shallow scans on our production environments. For that purpose it is common to configure and run a **Non-Destructive** scan configuration that is still as revealing as possible.



### Confident approach

Non-Credentialed surface <i>Exhaustive</i>	
Credentialed surface (Optionally) <i>Non-Destructive</i>	

If we take at face value our confidence in the target application and it's hosting environment, we rely on proper *Capcha* protection and such mechanisms that protects exposed surface from any state-altering - then we might scan even production or state-sensitive test environments with the full-blown **Exhaustive** feature set. Launching active probing for injection testing, fuzzing, active XSS tests, etc.

To be clear this treatment is something any production site *should hold up to*, on it's non-credentialed surface - certainly any internet exposed site. The question is just whether we want to validate this "boldly" by our own scans on the production environment (or if we leave it for someone else to do it [!]). In some scenarios we can combine this with running a **Non-destructive** scan with credentials.



### Intrepid approach

Non-Credentialed surface <i>Exhaustive</i>	
Credentialed surface <i>Exhaustive with explicit element exclusions</i>	

A more *intrepid* approach to data- and state sensitive environments is to scan even the Credentialed surface with an **Exhaustive** scan, in which we just make targeted and explicit exclusions to avoid particular Web forms, DOM elements, API endpoints or URL Paths, where we know that the site is vulnerable to offensive tests.

Note that this approach requires us to get notified beforehand if a new release of the app contains new paths, DOM elements or internal referenced APIs that can cause data corruption or state-change when scanned.

### Offensive approach

Non-Credentialed surface <i>Exhaustive</i>	
Credentialed surface <i>Exhaustive</i>	

To offensively scan everything with an **Exhaustive** scan, including the Credentialed surface without discretion, is usually an option only for test environments. And sometimes not even feasible for test environments. In case we need to share the environment with data dependent acceptance or release testing or if data re-staging is not easily done, then we can consider the Intrepid approach instead.

## B. Schema for classification: *Web Application Customization Spectrum*

A classification schema (1-5) for the degree of customization of an application with web interface, detected externally or internally and pertaining to any corporate environment.

### # Web App Customization Spectrum

#### ## 1. Off-the-Shelf SaaS

Web interfaces for **fully managed third-party SaaS solutions** with standardized functionality and limited configuration options.

**\*\*Examples:\*\*** *Microsoft 365 (Outlook Web, SharePoint Online), Salesforce, Workday, Google Workspace, Slack, Zoom*

#### ## 2. Configured Commercial Products

Commercially **licensed software products** deployed in-house or hosted by third parties, featuring organization-specific configuration and data but **no custom codebase**.

**\*\*Examples:\*\*** SAP ERP systems, Oracle E-Business Suite, Microsoft Dynamics, ServiceNow (standard implementation), Adobe Experience Manager (basic setup)

Admin interfaces on deployed appliances can also be regarded as a large subcategory under this class.

**Customization of:**

*deployment configuration, hosting, update management.*

#### ## 3. Extended Platform Solutions

Pre-built platforms or frameworks **extended through plugins, modules, or configurations that blend standard functionality with custom components**.

**\*\*Examples:\*\*** Extendable CMS systems such as WordPress, Drupal, Joomla with custom themes/plugins, Magento, Shopify Plus with custom extensions, Salesforce with custom or community contributed Lightning components.

**Customization of:**

*Contributed modules with limited custom application logic*

#### ## 4. Framework-Based Custom Applications

**Purpose-built applications** developed in-house or by contractors using established, up-to-date, and **actively maintained development frameworks** and following standard software development lifecycle processes.

**\*\*Examples:\*\*** *Applications built on Spring Boot, .NET Core, Django, Ruby on Rails,*

*React/Angular/Vue with backend APIs, Laravel PHP applications*

**Customization of:**

*All application logic, implemented with frameworks and supported by framework security updates.*

## **## 5. Framework-Less Custom and Legacy Apps**

Applications with web interfaces developed with bare-bone coding or based on no-longer-maintained frameworks.

**\*\*Examples:\*\*** *Applications using raw servlet APIs without frameworks, custom HTTP listeners, bespoke applications with web interfaces added as afterthoughts, inherited or custom built data processing tools with primitive archaic web UIs.*

**Customization of:**

All application logic - and *without* support from framework security updates.

May also be built with custom request handling, request dispatch logic and session handling.

Absence of active framework support will lead to greater need for custom coded data validation, sanitization and escape routines as well as maintenance of the underlying http framework mechanics.



## C. Expanding on *Web App Scanning vs. Network Vulnerability scanning*

This section examines some of the important unique capabilities that a dedicated DAST tool adds to a Vulnerability Management program. We will highlight key characteristics of DAST and address the specific gaps it fills beyond traditional host-based vulnerability scanning with tools like Nessus.

### C.1. Targeting specific apps by URL

Whereas Nessus and other broader vulnerability scanners may potentially scan all exposed services<sup>ii</sup> (if you allow it to) on an IP or against a range of IPs, a web application scan will typically target a single URL instead (or a limited enumeration of URLs).

For a certain host we fix not only the protocol (https) and hostname/IP, but also the port, the path and optionally query parameters to target a specific application.

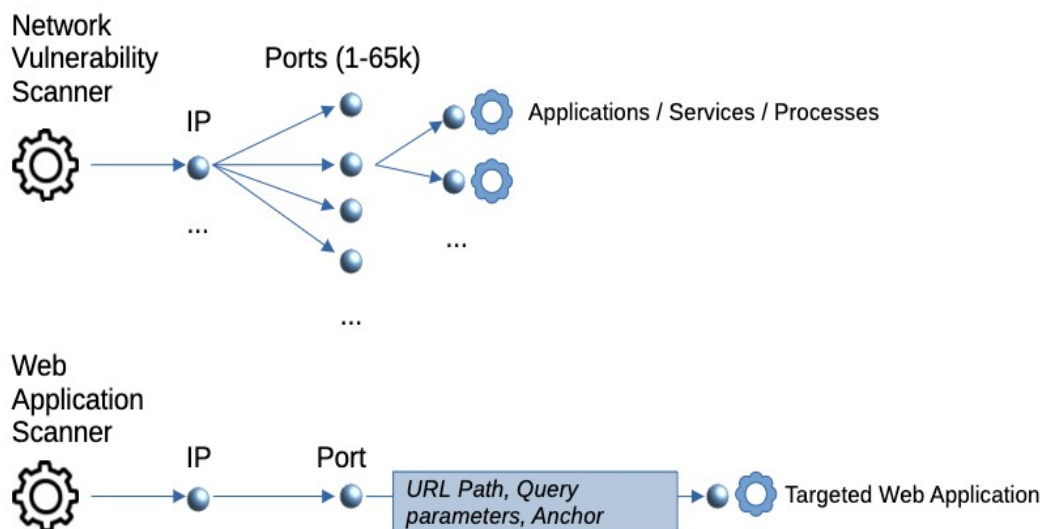


Figure 3:

1. General Network Vulnerability scan “spraying” an IP on all ports and services looping over: IP – Ports (0-65k) – all detected protocols.
2. The Web application Scanner (DAST). Locking in by: IP – Port (fixed) – Web Server – [Path(fixed), Optionally also: Query parameters and Query anchor] – Precisely targeted Web application

## C.2. DOM interactions and DOM aware scanning

Modern web application vulnerability scanning requires deep DOM (Document Object Model) interpretation capabilities<sup>iii</sup> to effectively analyze JavaScript-heavy applications, unlike traditional or more general-purpose scanners that may examine HTTP headers, SSL/TLS configuration and possibly *some* HTML/JS constructs such as more easily accessible inclusions and dependencies.

The dedicated web application scanner must build and maintain an in-memory representation of the web application's state by *executing* the JavaScript code, similar to how a browser renders a Single Page Application (SPA). While crawling the application, it must track DOM mutations and state changes, identifying interactive elements and potential attack surfaces that only become visible after JavaScript execution.

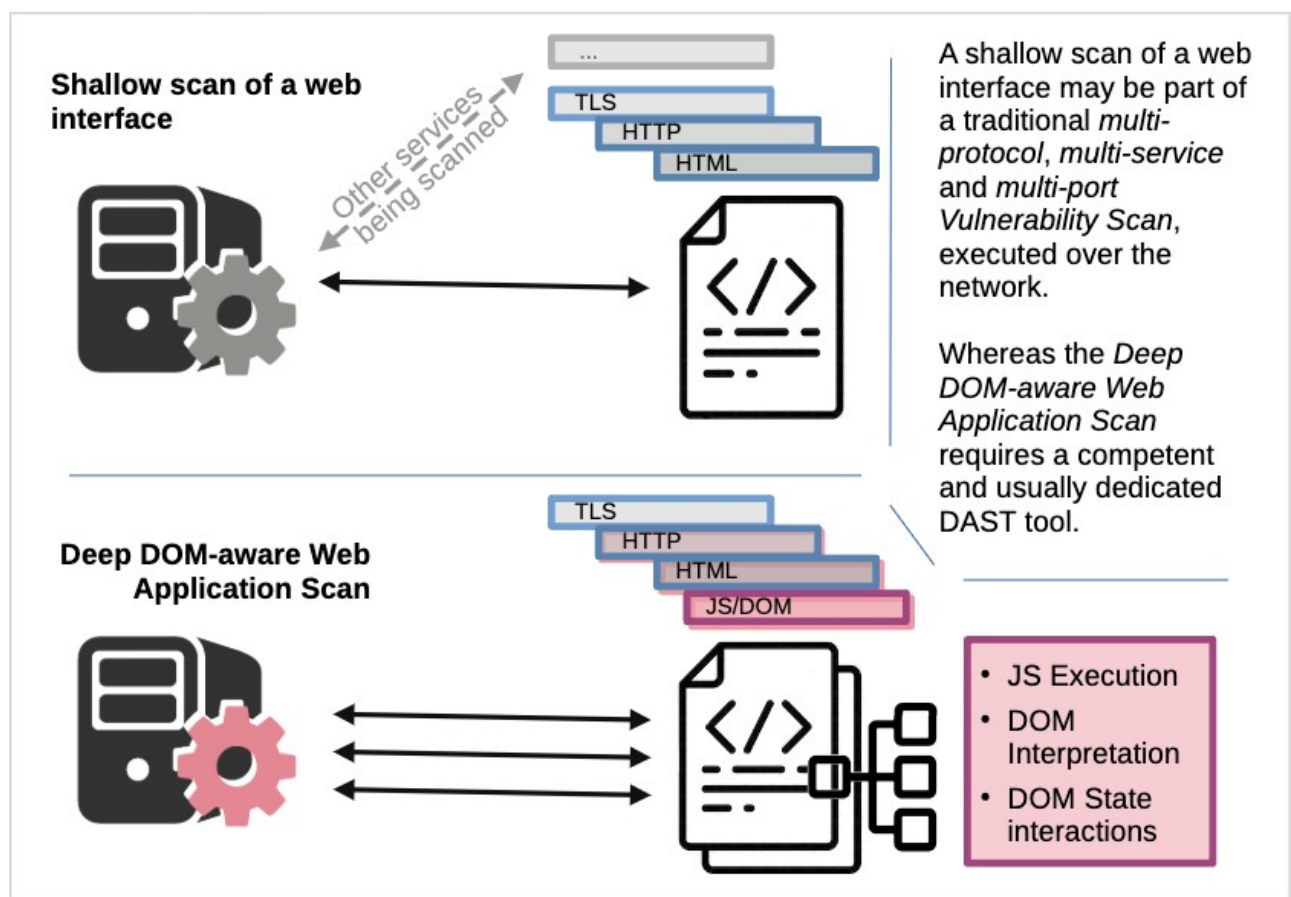


Figure 4: Traditional or general-purpose vulnerability scanners can usually detect basic security issues through TLS-level, HTTP-level and to varying extent HTML-level analysis but lack the sophisticated DOM building and parsing needed for comprehensive testing of modern web applications.

## C.3. Active Probing techniques

In the below and throughout this document we have used the terms **Active vs Passive Probing** to specify assessment methods that actively submits data into detected endpoints, in order to validate alterations in an application's state and flaws in it's responses. We avoid the frequently used term **Passive Scanning**, since this term often has a different meaning in the context of Vulnerability Management (as in *Passive Vulnerability Scanners*<sup>11</sup>).

More generalized scanning tools typically have some more passive probing techniques for Web Interfaces. They request the application index page and referred resources, evaluate TLS security posture, http headers and to varying extent html constructs and inclusions. They might even have some capabilities for crawling the application.

Active probing techniques involve launching test sequences with *web fuzzing*, where invalid, malformed, or unexpected inputs are systematically injected into the application to identify security vulnerabilities in both frontend and backend implementations<sup>12</sup>. This process specifically targets input/output handling mechanisms, including validation, sanitization, and escaping routines, to detect potential injection flaws and cross-site scripting (XSS) vulnerabilities.

Particularly important for in-house developed applications / Custom codebases

Active probing techniques becomes **particularly critical for applications with extensive custom code and in-house implementations**, as these unique codebases often contain application-specific vulnerabilities that other standard testing techniques (such as [those based on framework and library version recognition](#)) will miss.



The presence of custom codebase is also one of the key natural prioritization criteria (as discussed in section [Application prioritization](#)).

Since the more Active Probing techniques also tend to be the more *offensive* ones (see: [Active Probing techniques: Offensiveness](#)), this emphasizes the importance of access to test environments.

It can be tempting to equate *Active* or *Offensive* Web Application testing as analogous to usage of state-changing HTTP methods such as POST, PUT, DELETE (as opposed to “reading” or “safe” methods like GET and HEAD)<sup>13</sup>. However this would be an oversimplification only valid in an idealized and non-adversarial scenario, since a DAST tool (and other abusive clients) can certainly embed destructive payloads also in GET parameters or headers. True offensiveness depends on payload intent so refer to vendor recommendations and guidelines to understand what scan configurations are generally safer (e.g. when scanning production environments) and what configurations, modes, or settings will be more Active/Offensive. See also [D: Unpacking Active vs. Passive probing](#).

<sup>11</sup> <https://attaxion.com/blog/active-and-passive-vulnerability-scanning-what-is-the-difference/>  
<https://www.tenable.com/solution-briefs/nessus-network-monitor>

<sup>12</sup> <https://www.techtarget.com/searchsecurity/tip/Web-fuzzing-Everything-you-need-to-know>  
<https://www.packetlabs.net/posts/what-is-fuzzing/>

<sup>13</sup> RFC 7231 explicitly warns against state-changing GET requests, yet many legacy APIs violate this standard.  
<https://datatracker.ietf.org/doc/html/rfc7231#section-4.2.1>. And of course, from an adversary perspective such standards are to be violated.

## Active Probing techniques: Offensiveness

Exhaustive scans with active probing or exploitation testing should be conducted in a controlled manner that minimizes risk to business operations.

Running comprehensive DAST scans in production during business hours presents significant risks:

- **Exposing or corrupting production data.** DAST tests that manipulate data to check application behavior could also inadvertently expose or corrupt sensitive production data.
- **Creating garbage data.** Form submission by the scan can create garbage in databases and trigger emails by submitting contact forms etc. This is of course particularly true for credentialed Web Applications scans since we usually expect our forms to be either protected with login or with captcha.
- **Inadvertent DOS effects.** Active scanning can overwhelm servers with requests and impact legitimate user experiences. Even if deliberate DOS or load testing is not the purpose, an extensive web application scan can have such a side effect, particularly of course on not well scaled environments.

**Remark:** Not only active and offensive probing can have inadvertent DOS effects. For any scan that is crawling the target site and launching a high volume of requests with high concurrency this can be an issue to look out for (can be more prone to appear when scanning internal environments, deployed in more basic infrastructures).

A competent DAST tool typically has a configurable spectrum of scan configuration options where the most quick and shallow scans are no more intrusive than loading the site in a browser, and the most extensive and deep scans can go on for hours, crawling<sup>12</sup> the target site and submitting thousands of requests, many times unsafe ones.

Some good practices<sup>iv</sup> are:

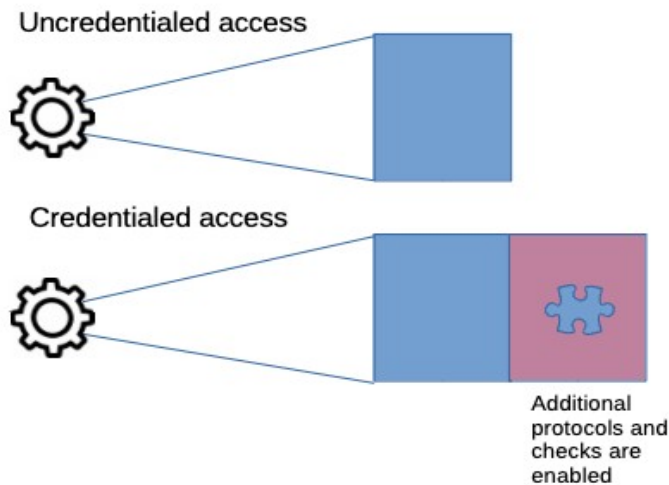
- Testing in a staging/QA environment first before moving to production.
- Starting with smaller, less intrusive scans before scaling to more comprehensive testing.
- Using rate limiting and throttling to prevent overwhelming the servers with too many requests.
- When production testing is implemented, it should be carefully tailored and scheduled during off-hours to minimize business impact. An exception from this rule of caution can be the most quick and shallow scans (such scans that do not submit any data and do not extensively crawl the application).

## C.4. Credentialed scanning with a DAST tool

### Increasing *surface coverage* rather than extending the scope of protocols and checks

An important part of the DAST tool feature set is the ability to implement credentialed scanning. There are however some key differences in how to look at credentialed scans of web applications versus credentialed host scans.

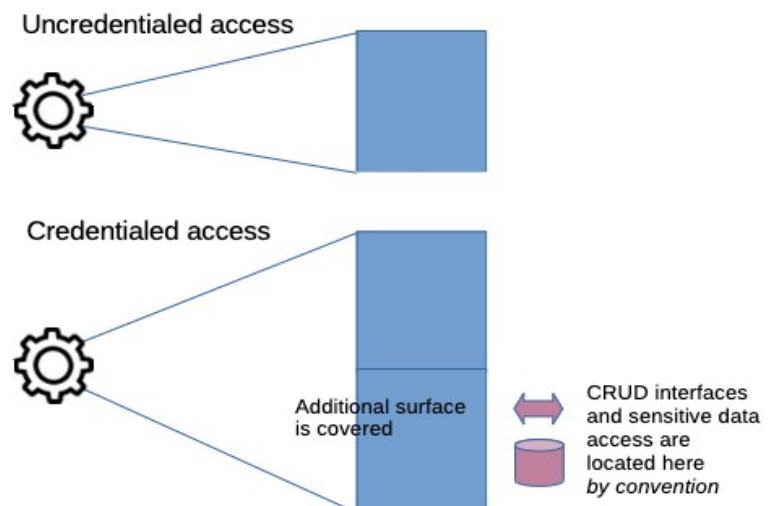
#### Scanning a host over the network



When conducting a credentialed network scan with a product such as Nessus, giving the scan credentialed access will **open up new protocols** and internal APIs such as *WMI, Windows Registry, File system, Linux Package managers*, etc. This will enable *an additional set of vulnerability plugins or assessments to execute* ([link](#)) as compared to running the scan without credentials.

#### Scanning a web application

A credentialed scan against a web application on the other hand, will not typically enable *other* plugins or vulnerability checks to execute, rather it will **increase the surface coverage** with the *same set* of plugins or checks. We *expect* application admin, CRUD interfaces and sensitive data to be behind credentialed access, but this is *by convention and not by hard protocol* restrictions. Ultimately, this is up to the designer of the web application.



## Toolkit rather than ascertained feature support

In traditional network scanning, to enable machine access to a host OS, traditional service accounts can be used. Login mechanisms such as SSH, SMB and SNMP are well suited for automated credentialed access to be established. The scanner implementation does not need to invent the wheel or come up with hacks in order to establish a remote session.

For a web application however, *if* the developer has intended to enable machine login for administrative or integration purposes, then the method for this enablement is to expose an API. And API scanning can certainly be included in your web application scans.

Mere API access however, even if available, does not satisfy the access requirement in order to do a *full Web UI vulnerability assessment*. Instead we can make the scanner *mimic an interactive user session and attempt to login via the user interface*. This is **abusive by nature**, as the web UI is not intended to support this purpose but on the contrary it may have mechanisms attempting to obstruct and impede replayed and automated logins. This poses a challenge to the DAST tool vendor as well as the user. A toolkit is hence provided to overcome this, and that toolkit often contains several different methods to establish and maintain an authenticated session. Thus, to implement credentialed web application scanning often becomes a process of trial and error.

In the end there is no guarantee for credentialed access without manipulation of the site's configuration. If credentialed Web UI scanning is a hard requirement, this *may* ultimately lead to the necessity to deploy the app with special configuration in a test environment, where certain login and session protection mechanisms can be disabled for the purpose of enabling credentialed scanning.



A flow-chart like procedure for implementing credentialed scanning can be built with steps like this:



1. *ASSERT IF*: Login is supported on http level [Basic, NTLM, Kerberos]
2. *IF NOT* – Check if automated login form submission can be done
3. *IF NOT* – Check if a login *recording-and-replay* can be implemented (typically with Selenium script)
4. *IF NONE* of the above *OR IF MFA* – Check if Session hijacking (Cookie theft) can be implemented
5. *IF NONE* of the above approaches work – Then work with the developers – special deployment configuration for testing might be necessary.

Ultimately, necessary disablement to make possible credentialed scanning of a test environment can be:

- To remove MFA
- To allow permanent or long-lasting sessions
- To disable mechanisms for identifying and denying suspicious clients/user agents
- To disable mechanisms for identifying and killing shared sessions between user agents
- To allow-list user-agent/scanner IPs

## C.5. Ideal Scan cycle implementation may differ

**Universal network and host-based vulnerability scans are typically scheduled in regular scan intervals — such as daily, weekly, or bi-weekly —** Sometimes with critical systems scanned more frequently<sup>14</sup>. These scheduled scans can be coordinated with other operational functions, particularly Patch Management cycles, and may be aligned with calendars such as Microsoft's Patch Tuesday release calendar (occurring on the second Tuesday of each month)<sup>15</sup>. Some teams implement flexible scheduling that triggers immediate scans when significant system changes occur (builds, installations, deployments, etc.) or when alerted on new emerging threats.

**Web application scans**, which can vary in depth and intensity, can be incorporated into the regular scheduled scan cycle of the Vulnerability Management. However, when implementing in-depth DAST scanning in response to in-house application development, then these scans are *ideally coordinated with the development team's release cycle*<sup>16</sup>.



This approach allows for thorough scanning of release candidates in test or staging environments before approval. Optionally or additionally, build pipelines can be configured to automatically trigger scans of new application builds, aligning the security testing with the development process<sup>17</sup>.

<sup>14</sup> Best practices for creating effective scan schedules in Invicti Enterprise <https://www.invicti.com/support/scheduled-scans-best-practices/>

Best Practices for Vulnerability Scanning - Scytale <https://scytale.ai/resources/best-practices-for-vulnerability-scanning-when-and-how-often-to-perform/>

<sup>15</sup> 6 Steps of Vulnerability Scanning: Best Practices - RedLegg <https://www.redlegg.com/blog/6-steps-of-vulnerability-scanning-best-practices>

<sup>16</sup> What is Dynamic Application Security Testing (DAST) - OpenText <https://www.opentext.com/what-is/dast>

Continuous Dynamic Application Security Testing (DAST) - Black Duck <https://www.blackduck.com/dast.html>

<sup>17</sup> DAST: Web App & API Vulnerability Scanning - Cobalt <https://www.cobalt.io/platform/dast>

## C.6. Specialized Focus: Vast framework and library recognition

Unlike multi-purpose network scanners that offer certain limited web vulnerability detection capabilities, dedicated DAST tools provide specialization in web app security and a development roadmap that prioritizes staying up-to-date with vulnerabilities, exploits and detection techniques in the Web Application landscape.

A big important and usually Non-offensive subset of detection techniques is recognition and version enumeration among a plethora of web frameworks and included libraries<sup>v</sup>, and mapping those to known vulnerabilities. The diversity in the web framework and JS dependency landscape necessitates that effective DAST tools maintain comprehensive, up-to-date detection capabilities for a plethora of framework-specific vulnerabilities. One of the value propositions of specialized DAST solutions lies in this ability.

Because of the product focus with *vast framework and library recognition*, the introduction of a DAST tool can provide a significantly increased visibility, complementing that of your general network scans. Even in those cases where we are restricted from employing more [active probing techniques](#) and we have to resort to [non-destructive scans](#). That may be e.g. in scenarios when we only scan our production environments, as a complement to general network scanning on those same hosts.



### **Tenable implementation**

See the largest of the plugin families in the WAS category: *Component Vulnerability*.

<https://www.tenable.com/plugins/was/families?type=was>

This plugin family contains identification of a vast range of *inclusions, libraries and dependencies*, and keeps up-to-date with their vulnerabilities.



## D: Unpacking *Active vs. Passive probing*

Like we discussed in *C.3. Active Probing techniques* and *Active Probing techniques: Offensiveness* the distinction between what is Active and not Active is not always clear-cut, it is not as simple as classifying for example by what *http* methods are invoked in a test. Various vendors might also use their own concepts and terminology, such as *Non-offensiveness*, *Non-destructive*, *Safe Mode*, etc. Also, a functionally non-destructive scan can have a *DOS-like impact* on a target site, particularly this may be the case on internally deployed applications, that are not always well dimensioned to handle a flood of requests.

Conclusively, it may be better to look at this as a sliding scale (*more-or-less Active probing*) rather than as a rigid classification where there is a common consensus. We can still attempt to outline some commonly understood typical traits of Active vs Passive probing in Web Application Scanning:

### Active Probing

Active probing involves direct interaction with the application through <sup>18</sup>:

- *Submitting forms and data to endpoints*
- *Injecting attack payloads into HTTP requests*
- *Testing input validation and sanitization*
- *Fuzzing parameters and API endpoints*
- *Analyzing application responses to malformed inputs*

The key characteristic is that active scanning/probing attempts to identify vulnerabilities by actually sending potentially malicious requests and analyzing how the application handles them<sup>19,20</sup>.

### Passive Probing

Passive probing examines the application without making potentially disruptive requests<sup>21</sup>:

- *Analyzing HTTP headers and security policies*
- *Evaluating TLS/SSL configuration*
- *Examining dependencies to standard components and libraries*
- *Examining exposed tokens or sensitive information*

---

18 [https://docs.gitlab.com/ee/user/application\\_security/dast/browser/](https://docs.gitlab.com/ee/user/application_security/dast/browser/)

<https://www.jc-cybersecurity.co.uk/what-is-web-application-security-testing-and-how-can-it-help>

19 DAST browser-based analyzer - GitLab Documentation [https://docs.gitlab.com/ee/user/application\\_security/dast/browser/](https://docs.gitlab.com/ee/user/application_security/dast/browser/)

20 Vulnerability Scanners: Passive Scanning vs. Active Scanning <https://www.zengrc.com/blog/vulnerability-scanners-passive-scanning-vs-active-scanning/>

21 What OWASP ZAP can do, and when to use it - Jit.io <https://www.jit.io/resources/owasp-zap>

- *Reviewing content security policies*
- *Inspecting HTML structure and client-side code*

## Key Distinctions

### ***Impact and Risk:***

- Active probing can potentially disrupt services or affect application state<sup>22,23</sup>
- Passive probing is non-intrusive and safe for production environments<sup>24</sup>

### ***Detection Capabilities:***

- Active scanning finds more vulnerabilities but requires careful execution
- Passive scanning is limited to surface-level detectable issues but can run continuously<sup>25</sup>

### ***Usage Context:***

- Active probing is best suited for testing environments where potential disruption is acceptable<sup>26</sup>
- Passive probing can be safely performed in production, providing continuous monitoring

Both techniques complement each other and are typically used together in comprehensive web application security testing strategies.

---

22 Active vs Passive IAST Scanning - Contrast Security <https://www.contrastsecurity.com/glossary/active-vs-passive-iaast>

23 DAST vs Manual Pentesting vs Automated Pentesting: 5 Differences <https://www.cycognito.com/learn/application-security/dast-vs-manual-pentesting-vs-automated-pentesting.php>

24 Dynamic Application Security Testing (DAST): How Safe is Your ... <https://codenteam.com/dynamic-application-security-testing-dast-how-safe-is-your-application-in-action/>

25 Vulnerability Scanners: Passive Scanning vs. Active Scanning <https://www.zengrc.com/blog/vulnerability-scanners-passive-scanning-vs-active-scanning/>

26 Advantages and Disadvantages of Active vs. Passive Scanning in IT ... <https://www.infosecurity-magazine.com/opinions/active-passive-scanning/>

## E: Expanding on *Common functions and perspectives on Web Application Scanning*

### **Vulnerability Management**

Organizational function or team for *Vulnerability Management*.

The *Vulnerability Management Program* encompasses the entire organizational framework for managing security vulnerabilities across systems, applications, and infrastructure. The program follows a continuous lifecycle of discovering, prioritizing, assessing, remediating, and validating security vulnerabilities through systematic processes, supported by policies, tools, and defined roles and responsibilities.

### **Dev team (or SDLC aligned QA function)**

Scanning of Web Applications as part of a broader spectrum of Application Security practices employed as part of *Application development*.

Some typical such practices, that can be more or less owned by the dev team itself are:

- Dynamic Application Security Testing (DAST)
- Static Application Security Testing (SAST)
- Software Component Analysis (SCA)
- Verification of Security Controls and Compliance (OWASP ASVS etc)
- Verification of secure coding practices (manual such as Peer-Review and automated as part of SAST)
- Security Architecture (Design and deployment of all relevant security layers spanning from outer layers such as WAFs via supporting Identity Federations, PKI,..etc down to inner workings such as *User challenge workflows*).

### **External Auditor or Pen Tester**

Independent security assessment of web applications as part of structured penetration testing and security audits.

The process typically includes pre-audit consultation, vulnerability identification, exploitation attempts, and detailed reporting with actionable recommendations<sup>27</sup>. Key activities encompass:

- Comprehensive security testing of applications, APIs, and related infrastructure
- Evaluation of security controls and their effectiveness
- Verification of compliance with industry frameworks and regulations
- Discovery and analysis of security weaknesses in design and implementation
- Documentation of findings with detailed remediation strategies<sup>28 29</sup>

<sup>27</sup> <https://dsecure.me/en/services/what-is-an-it-security-audit/web-application-security-audit/>

<sup>28</sup> <https://www.blazeinfosec.com/post/web-application-penetration-testing/>

<sup>29</sup> <https://www.dungeondata.com/en/blog/web-application-audit-pentest/>

## End notes

i References:

<https://www.blazeinfosec.com/post/web-application-penetration-testing/>  
<https://www.getastra.com/blog/security-audit/security-audits/>

- ii We illustrate the general case of port/application mapping with a *one-to-many* relationship. In typical scenarios, applications are mapped to ports in a *one-to-one* relationship. However, multiple applications or processes may concurrently bind to the same port on the same IP address using the `SO_REUSEPORT` socket option. This further highlights the ambiguity regarding which applications or processes are being targeted during a general-purpose network scan.

References:

- [1] Binding two processes on the same port for fun and firewall evasion <https://github.com/sghctoma/multipass>  
[2] `SO_REUSEPORT` on linux - sockets - Stack Overflow <https://stackoverflow.com/questions/3261965/so-reuseport-on-linux>

iii References:

1. [MDN: Introduction to the DOM]([https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction))  
2. [PortSwigger: DOM-based Vulnerabilities](<https://portswigger.net/web-security/dom-based>)  
3. [BrowserStack: DOM in Web Automation](<https://www.browserstack.com/guide/dom-in-selenium>)

DAST-specific DOM interpretation details: [Acunetix DAST Guide](<https://www.acunetix.com/the-ultimate-guide-to-dast/>) and [OWASP Testing Guide](<https://www.imf.org/external/pubs/ft/bop/2007/pdf/bpm6.pdf>)

iv References on production safety of DAST:

<https://www.invicti.com/blog/web-security/how-to-implement-dast-effectively-guide-and-tips/>  
<https://research.aimultiple.com/dast-best-practices/>  
<https://www.blackduck.com/blog/production-safe-dast.html>  
[https://www.reddit.com/r/blueteamsec/comments/gd8c48/q\\_running\\_dast\\_on\\_production\\_or\\_development/](https://www.reddit.com/r/blueteamsec/comments/gd8c48/q_running_dast_on_production_or_development/)  
<https://qualysec.com/dynamic-application-security-testing/>  
<https://brightsec.com/blog/why-running-dast-in-production-is-not-a-good-idea/>

- v For instance, the PHP ecosystem boasts a very diverse and extensive array of web frameworks, (even to greater extent than for Java, Python, C# or Ruby). PHP's landscape is characterized by a very fragmented and varied selection including Laravel, CodeIgniter, Symfony, Laminas (formerly Zend), Phalcon, CakePHP, and Yii, among others[1][4][7]. Each catering to different development needs, from full-stack solutions to microframeworks. It also includes CMS platforms such as Drupal, WordPress and Joomla.

Similarly, the JavaScript ecosystem presents a vast and rapidly evolving landscape dominated by frameworks and libraries such as Angular, React, Vue.js, jQuery, and numerous emerging technologies.

Sources

- [1] 11 Best PHP Frameworks For Beginner to Pro Developers - Hostinger <https://www.hostinger.com/tutorials/best-php-framework>  
[2] Java vs PHP vs Python | What are the differences? - StackShare <https://stackshare.io/stackups/java-vs-php-vs-python>  
[3] PHP vs. Other Frameworks: Choosing the Best for Website ... <https://tech.eastsons.com/blog/php-vs-other-frameworks-choosing-the-best-for-website-development>  
[4] Top 5 Php Frameworks to be included in the Best of 2025 List <https://blog.oloma.dev/top-5-php-frameworks-to-be-included-in-the-best-of-2025-list-9f56c181514d>  
[5] PHP vs Python: A Comparison Between the Two Languages - Kinsta <https://kinsta.com/blog/php-vs-python/>  
[6] PHP Frameworks (CodeIgniter, Yii, CakePHP) vs. Django [closed] <https://stackoverflow.com/questions/2578540/php-frameworks-codeigniter-yii-cakephp-vs-django>  
[7] Los 8 Mejores Frameworks PHP para Desarrolladores Web <https://www.hostinger.es/tutoriales/mejores-frameworks-php>