

Ataques web (y vulnerabilidades que los permiten)

OWASP

OWASP (<https://www.owasp.org> ; Open Web Application Security Project) és una organització internacional sense ànim de lucre que desenvolupa guies, eines i protocols d'actuació que qualsevol entitat/empresa pot aplicar en diferents àmbits de la seguretat informàtica (no només web). Entre les seves publicacions podem trobar:

*La llista de les "**Top 10**" vulnerabilitats (o problemes de seguretat) web més importants/comunes. (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project) Hi ha hagut varies actualitzacions d'aquesta llista al llarg dels anys; l'actual és del 2017 i es pot consultar també en format PDF. L'objectiu és formar i conscienciar a tots els implicats en la seguretat per reduir els seus riscos i conseqüències.

NOTA: Aquesta llista "Top 10" es basa en altres documents fonamentals, entre els quals podem destacar:

*La guia de proves OWASP (https://www.owasp.org/index.php/OWASP_Testing_Project)

*La guia de revisió de codi OWASP (https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project)

*La guia de verificació (https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project)

D'altra banda, a <https://www.owasp.org/index.php/Category:Vulnerability> es troba un recull oficial més complet i exhaustiu dels diferents tipus de vulnerabilitats web existents i les maneres d'evitar-les/protegir-les/eliminar-les. A més, a <https://github.com/OWASP/CheatSheetSeries> es poden trobar "xuletes" de les tècniques i bones pràctiques més comunes.

***Projectes software.** Tant de **defensa** (és a dir, un conjunt d'eines, llibreries i tècniques que permeten detectar i contrarestar atacs del nivell d'aplicació; https://www.owasp.org/index.php/Category:OWASP_Defenders) com d'**atac** (és a dir, un conjunt d'eines i tècniques que permeten realitzar proves de "pentesting"; https://www.owasp.org/index.php/Category:OWASP_Breakers) com de **desenvolupament** (és a dir, un conjunt d'eines, llibreries i tècniques que permeten desenvolupar aplicacions de forma segura en múltiples llenguatges i plataformes; https://www.owasp.org/index.php/Category:OWASP_Builders).

Com que la quantitat de projectes de defensa/atac/desenvolupament és molt gran, a l'enllaç https://www.owasp.org/index.php/Category:OWASP_Project#Project_Inventory es poden trobar classificats per ordre de rellevància i d'actualitzacions més freqüents els projectes més importants. Entre ells, podem destacar:

***Zed Attack Proxy** (<https://github.com/zaproxy/zaproxy>) Programa d'atac web que incorpora diferents mòduls ("web spider", escanejador de vulnerabilitats, HTTP MITM, etc).

***OWTF** (<https://github.com/owtf/owtf>) Una alternativa a ZAP -però amb interfície web- que segueix els protocols d'actuació i vulnerabilitats desenvolupades a la documentació generada per OWASP així com els estàndards NIST i PTES

***Nettacker** (<https://github.com/zdresearch/OWASP-Nettacker>) Eina OSINT que permet descobrir nodes a la xarxa i recopilar informació sobre ells (serveis que executen, configuracions -mal- establertes, vulnerabilitats trobades, etc) per tal de generar un informe.

***Web Testing Environment Project** (WTE; <https://github.com/mtesauro/owasp-wte>). És un conjunt seleccionat d'aplicacions de seguretat d'atac llestes per instal·lar de cop gràcies a un repositori mantingut per OWASP. Entre aquestes aplicacions podem trobar software com ara Nmap, Netcat, Nikto, Firefox, BurpSuite, Fierce, SQLmap, Wireshark, W3af, Wpscan, etc.

***ModSecurity CSR** (<https://coreruleset.org>) . Conjunt de regles prefabricades llestes per incorporar ModSecurity (<https://www.modsecurity.org>), un tallafocs especialitzat en tràfic HTTP/S (també anomenats genèricament "WAF", de "Web Application Firewall")

***Security Shepherd** (<https://github.com/OWASP/SecurityShepherd>). Aplicació web/mòbil vulnerable, especialment dissenyada per practicar habilitats de "pentesting". D'altra banda, hi ha moltes més plataformes (en aquest cas de tipus web o bé sistemes complets) expressament vulnerables pensats per poder-hi explotar vulnerabilitats de forma legal, com per exemple:

***WebGoat** (<https://github.com/WebGoat>)

***JuiceShop** (<https://github.com/bkimminich/juice-shop>)

***DVWA** (<https://github.com/ethicalhack3r/DVWA>)

***AltoroJ** (<https://github.com/AppSecDev/AltoroJ>)

***OWASP-VWAD** (<https://github.com/OWASP/OWASP-VWAD>) -És un recull-

Tipus de vulnerabilitats (web) més habituals

Encara que al recull OWASP de tipus de vulnerabilitats comentat als paràgrafs anteriors apareixen molts més, a continuació es mencionaran els tipus de vulnerabilitats més coneguts (de natura web, ja que en general són les més exposades), molts dels quals també apareixen al "Top 10" (aquests tipus de vulnerabilitats estan associats a diferents tipus d'atacs que es poden veure llistats a <https://www.owasp.org/index.php/Category:Attack>):

***Cross-Site Scripting (XSS)** ([https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))): Permite, utilizando los parámetros de entrada de la aplicación, modificar y añadir código Javascript a la misma. Son vulnerabilidades que se encuentran en el servidor, pero que están destinadas a atacar al cliente: generalmente, se necesita que este siga un enlace de la aplicación, en el que se ha modificado algún parámetro, para permitir añadir código, que se ejecutará en el navegador del cliente. Normalmente la intención será un robo de cookies del cliente, predicción del id de sesión, etc. El cliente verá que al seguir el enlace llega realmente al sitio web que quería: no hay suplantación del mismo. Podemos agrupar los XSS en dos grupos:

**XSS reflejados*: El código modificado se elimina al cargar la página de nuevo. Está basado en la URL y, por tanto, al recargar la página, se elimina el mismo.

**XSS persistentes*: El código modificado queda almacenado en la web.

La forma de corregir esta vulnerabilidad es filtrar y validar todas las entradas de la aplicación. No se debe utilizar nunca una variable que se recibe desde el cliente, confiando en que ésta tendrá un valor correcto. Los lenguajes de programación incluyen diferentes funciones que permiten filtrar el contenido de las variables, dependiendo de dónde las vayamos a utilizar. Para ver más en concreto cómo se podría hacer esto en un servidor PHP, se puede consultar por ejemplo

<https://diego.com.es/ataques-xss-cross-site-scripting-en-php> o también

<https://ctf101.org/web-exploitation/cross-site-scripting/what-is-cross-site-scripting> o también

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md

***Cross Site Request/Reference Forgery (CSRF)** ([https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))): Esta vulnerabilidad es una evolución de los XSS donde un cliente anómalo se hace pasar por un cliente legítimo (utilizando datos parciales del mismo) para "robarle" la sesión. Esta vulnerabilidad está presente en formularios. Así pues, cuando estos se envían al servidor, es necesario asegurarse que la petición es legítima y debemos asegurarnos que el cliente ha realizado la petición, realizando los pasos previos necesarios. La forma más común para eliminar esta vulnerabilidad o, al menos, mitigarla, es la inclusión de tokens dinámicos. En los actuales Frameworks, suelen incluirse mecanismos para añadir esta protección a los formularios, de una forma muy sencilla. En algunos, por ejemplo, Laravel (utilizado para desarrollo de aplicaciones en PHP), basta añadir una etiqueta a la plantilla del formulario, para que se añada al mismo el token anti-csrf. Un ejemplo muy básico de en qué consiste se puede ver en <https://ctf101.org/web-exploitation/cross-site-request-forgery/what-is-cross-site-request-forgery/>

NOTA: Existen diferentes técnicas XSS según el lugar donde se consiga inyectar el código deseado. Algunas de ellas son: DOM Cross Site Scripting (DOM XSS), Cross Site Flashing (XSF), Cross Frame Scripting (XFS), Cross Zone Scripting (XZS), Cross Agent Scripting (XAS), Cross Referer Scripting (XRS), etc

***Local File Inclusion (LFI):** Estos tipos de ataques se aprovechan de alguna vulnerabilidad concreta en el servidor para poder subir en él ficheros (normalmente binarios) y así ejecutarlos desde el propio servidor. De esta manera se podría ejecutar, por ejemplo, un visor de logs, o un servidor ssh/consola Meterpreter, o un simple descargador de otros programas más avanzados (alojados remotamente en un servidor propiedad del "hacker" de tipo FTP o HTTP), etc. Un ejemplo de código PHP vulnerable se puede encontrar aquí: https://en.wikipedia.org/wiki/File_inclusion_vulnerability

NOTA: Even without the ability to upload and execute code, a LFI attack can be done because an attacker can still perform a [Directory Traversal / Path Traversal walk](#) using it as a LFI attack as follows: <http://example.com/?file=../../../../etc/passwd> In this example, an attacker can get the contents of the "/etc/passwd" file that contains a list of users on the server. Similarly, an attacker may leverage the Directory Traversal vulnerability to access log files (for example, Apache access.log or error.log), source code, and other sensitive information. This information may then be used to advance an attack. To prevent this, you should effectively filter any user input. More information in <https://www.acunetix.com/websitesecurity/directory-traversal/> An example of this attack can be seen here: <https://ctf101.org/web-exploitation/directory-traversal/what-is-directory-traversal/>

***Command Injection** (https://www.owasp.org/index.php/Command_injection): Estos tipos de ataques tienen como objetivo la ejecución de comandos arbitrarios en el sistema operativo host a través de una aplicación vulnerable. Estos ataques son posibles cuando una aplicación pasa datos inseguros proporcionados por el usuario sin una debida validación (formularios, cookies, encabezados HTTP, etc.) a un shell del sistema (por tanto, los comandos del sistema operativo proporcionados por el atacante generalmente se ejecutan con los privilegios de la aplicación vulnerable). Por supuesto, existen herramientas que automatizan esto como por ejemplo, **Commix** (<https://github.com/commixproject/commix>) Ejemplos de aplicaciones vulnerables se pueden encontrar en https://www.owasp.org/index.php/Command_Injection y un ejemplo en <https://ctf101.org/web-exploitation/command-injection/what-is-command-injection>

***SQL Injection** (https://www.owasp.org/index.php/SQL_Injection): Este tipo de ataques permiten acceder y manipular la BBDD subyacente tras la aplicación web. La idea es modificar las consultas que hace la aplicación a la base de datos, aprovechando las entradas de usuario a la aplicación, para poder obtener y/o modificar datos que a priori no deberían poderse consultar/alterar. En este sentido, hay que tener en cuenta que los motores de BBDD, tienen sus propias tablas para almacenar metadatos (por ejemplo, las bases de datos que hay, qué tablas tiene cada BBDD, las columnas y tipo de cada una, información de usuarios, etc), las cuales serán el objetivo más goloso, generalmente. Un ejemplo es <https://ctf101.org/web-exploitation/sql-injection/what-is-sql-injection>

Como ejemplo, explicaremos cómo se realizaría un SQLi de tipo UNION. En este tipo de ataque lo primero que debe hacerse es negar la consulta original, para que esa no sea la que devuelva la información; por lo tanto, se debe añadir una condición que siempre sea falsa, así por ejemplo: *SELECT ID, Nombre FROM Clientes WHERE ID=1 AND 1=0*

A partir de aquí, comprobaremos cuantas columnas devuelve la consulta original. Esto es importante, porque los SQLi deben devolver exactamente el mismo número de columnas. Para ello, vamos probando mostrando un valor constante, luego dos, etc. hasta que veamos que tenemos un resultado.

```
http://dominio/index.php?id=1 AND 1=0 UNION SELECT 1
http://dominio/index.php?id=1 AND 1=0 UNION SELECT 1,2
...
```

Seguidamente habrá que intentar obtener información de los nombres de tablas que hay en la BBDD, luego de los nombres de columnas por cada tabla y, una vez tengamos esa información, podremos ir extrayendo aquello que necesitamos. Dependiendo del motor de BBDD que estemos atacando, podremos hacer uso de funciones predefinidas que lleve incorporadas; por ejemplo, en MySQL existen las funciones *user()* (que devuelve el usuario que ejecuta las consultas en la aplicación) o *database()* (que devuelve el nombre de la base de datos usada actualmente).

```
http://dominio/index.php?id=1+and+1=0+union+select+1,2,user(),database()
```

Otro escenario común es utilizar una consulta para validar la autenticación de un usuario. Se consulta en la BBDD si el nombre de usuario y password existen y, si devuelve un registro, entonces se le permite el acceso. Si alguna de estas variables es vulnerable a un SQLi, se puede forzar la consulta para que siempre devuelva algo, de manera que podremos saltar el login de la aplicación.

NOTA: Hay que tener presente que la SQL injection se hará con los permisos que disponga el usuario de la BBDD con el que se haya ejecutado la consulta en cuestión.

Por supuesto, existen herramientas que automatizan esto. Por ejemplo, **SQLMap** (<http://sqlmap.org>) , nos permite automatizar los SQLi y, una vez detectado un parámetro vulnerable, nos permite extraer información de una manera muy cómoda. Incluso nos permitirá abrir una shell en el servidor, si esto es posible con el usuario actual, subir ficheros, realizar elevación de privilegios, etc.

Para prevenir los SQLi, se deben parametrizar las consultas y es necesario filtrar y comprobar el valor de las entradas. También es muy importante restringir al máximo los permisos del usuario con el que la aplicación se conecta a la BBDD y, por supuesto, para cada BBDD, utilizar un usuario diferente. Esto es muy importante y, aunque es una mala práctica usar una misma instancia del motor de BBDD para diferentes aplicaciones, es un escenario muy común. Por este motivo, si un usuario de la BBDD tiene permisos sobre varias BBDD de diferentes aplicaciones y una de éstas presenta esta vulnerabilidad, el atacante podría obtener los datos del resto de aplicaciones. Además de esto, es muy importante ocultar los errores provocados por consultas en BBDD; esto es porque la forma de detectar un SQLi es intentar forzar un error (la típica ') y, una vez se comprueba que existe, si se muestra al usuario, éste puede extraer información, como el motor de BBDD usado, etc., con lo que se le facilitará la realización del ataque. Más consejos hay en

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md

NOTA: D'atacs "injection" n'hi ha de diferents tipus:

"LDAP Injection" (https://www.owasp.org/index.php/LDAP_injection)

"XPath Injection" (https://www.owasp.org/index.php/XPATH_Injection)

"Log Injection" (https://www.owasp.org/index.php/Log_Injection), etc

Para encontrar más recursos (libros, documentacion, "cheatsheets", herramientas, webs, cursos, laboratorios, etc) sobre cómo implementar estos ataques y más, se puede consultar <https://github.com/infoslack/awesome-web-hacking> . Por otro lado, en <https://capec.mitre.org> se puede ver una clasificación (muy interesante) por categorías de tipos de ataques. También se puede consultar <https://www.owasp.org/index.php?title=Category:Vulnerability>

Boletines de vulnerabilidades

Para estar al día de las diferentes vulnerabilidades que se van descubriendo se pueden consultar los siguientes servicios informativos:

<https://www.exploit-db.com> (<https://github.com/offensive-security/exploitdb>)

<http://cve.mitre.org/cve> ; <https://www.cvedetails.com>

<https://nvd.nist.gov> ; <https://www.us-cert.gov/ncas/alerts> ; <https://kb.cert.org/vuls>

<https://www.vulncode-db.com>

<https://vulldb.com>

<https://cxsecurity.com>

<https://seclists.org/bugtraq> ; <https://seclists.org/fulldisclosure>

<https://packetstormsecurity.com>

<https://www.securityfocus.com/bid>

<https://www.symantec.com/security-center/threats>

<https://wpvulndb.com> (especializado en Wordpress)

<https://securelist.com> (portal de noticias sobre seguridad)

NOTA: Els codis numèrics CVE ("Common Vulnerability and Exploit") que identifiquen moltes de les vulnerabilitats documentades són assignats per la Corporació Mitre, fundada per la National Cyber Security Division of the U.S. Homeland Security.