


Seguridad en Aplicaciones



OWASP

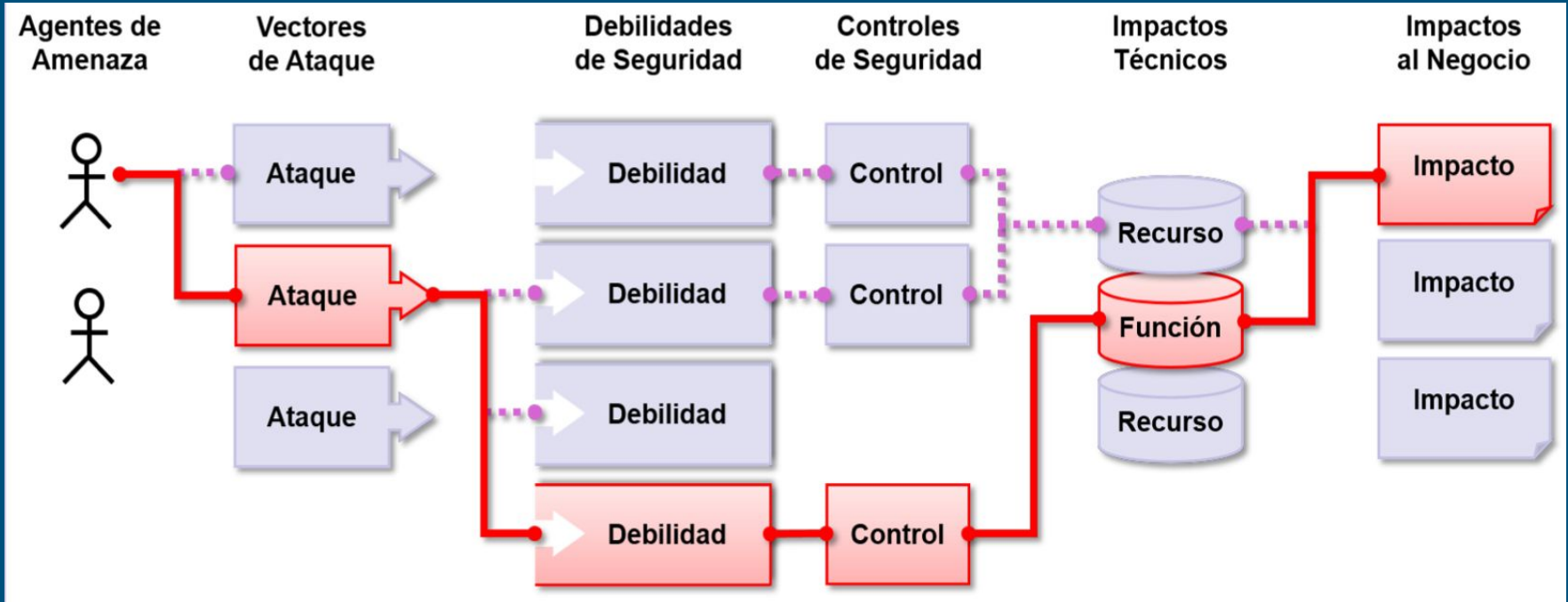


OWASP (Open Web Application Security Project)


Es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

<https://www.owasp.org/>

Riesgo en seguridad de Aplicaciones



Resumen riesgos top 10

Riesgo							Puntuación
		Explotabilidad	Prevalencia	Detectabilidad	Técnico	Negocio	
A1: 2017- Inyección	Específico de la Aplicación	FACIL: 3	COMUN: 2	FACIL: 3	GRAVE: 3	Específico de la Aplicación	8,0
A2: 2017 - Pérdida de Autenticación	Específico de la Aplicación	FACIL: 3	COMUN: 2	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	7,0
A3: 2017 - Exposición de Datos Sensibles	Específico de la Aplicación	PROMEDIO: 2	DIFUNDIDO: 3	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	7,0
A4: 2017 - Entidad Externa de XML (XXE)	Específico de la Aplicación	PROMEDIO: 2	COMUN: 2	FACIL: 3	GRAVE: 3	Específico de la Aplicación	7,0
A5: 2017 - Pérdida de Control de Acceso	Específico de la Aplicación	PROMEDIO: 2	COMUN: 2	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	6,0
A6: 2017 - Configuración de Seguridad Incorrecta	Específico de la Aplicación	FACIL: 3	DIFUNDIDO: 3	FACIL: 3	MODERADO: 2	Específico de la Aplicación	6,0
A7: 2017 - Secuencia de Comandos en Sitios Cruzados (XSS)	Específico de la Aplicación	FACIL: 3	DIFUNDIDO: 3	FACIL: 3	MODERADO: 2	Específico de la Aplicación	6,0
A8: 2017 - Deserialización Insegura	Específico de la Aplicación	DIFICIL: 1	COMUN: 2	PROMEDIO: 2	GRAVE: 3	Específico de la Aplicación	5,0
A9: 2017 - Componentes con Vulnerabilidades Conocidas	Específico de la Aplicación	PROMEDIO: 2	DIFUNDIDO: 3	PROMEDIO: 2	MODERADO: 2	Específico de la Aplicación	4,7
A10: 2017 - Registro y Monitoreo Insuficientes	Específico de la Aplicación	PROMEDIO: 2	DIFUNDIDO: 3	DIFICIL: 1	MODERADO: 2	Específico de la Aplicación	4,0

A1:2017 Inyección

Las fallas de inyección, como SQL, NoSQL, OS o LDAP ocurren cuando se envían datos no confiables a un intérprete, como parte de un comando o consulta. Los datos dañinos del atacante pueden engañar al intérprete para que ejecute comandos involuntarios o acceda a los datos sin la debida autorización.

Una aplicación es vulnerable a ataques de este tipo cuando:

- Los datos suministrados por el usuario no son validados, filtrados o sanitizados por la aplicación.
- Se invocan consultas dinámicas o no parametrizadas, sin codificar los parámetros de forma acorde al contexto.
- Se utilizan datos dañinos dentro de los parámetros de búsqueda en consultas Object-Relational Mapping (ORM), para extraer registros adicionales sensibles.
- Los datos dañinos se usan directamente o se concatenan, de modo que el SQL o comando resultante contiene datos y estructuras con consultas dinámicas, comandos o procedimientos almacenados.

Ejemplos de escenarios de ataque

Escenario #1: la aplicación utiliza datos no confiables en la construcción del siguiente comando SQL vulnerable:
`String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + "";`

Escenario #2: la confianza total de una aplicación en su framework puede resultar en consultas que aún son vulnerables a inyección, por ejemplo, Hibernate Query Language (HQL):
`Query HQLQuery = session.createQuery("FROM accounts WHERE custID=" + request.getParameter("id") + "");`

En ambos casos, al atacante puede modificar el parámetro “id” en su navegador para enviar: ' or '1'='1. Por ejemplo:

`http://example.com/app/accountView?id=' or '1'='1`

Esto cambia el significado de ambas consultas, devolviendo todos los registros de la tabla “accounts”. Ataques más peligrosos podrían modificar los datos o incluso invocar procedimientos almacenados.

Ejemplos en NodeGoat

A1 - 1 Server Side JS Injection : (contributions.js 24) (fix: parseInt())

- `process.kill(process.pid)`
- `res.end(require('fs').readdirSync('.').toString())` `res.end(require('fs').readFileSync('server.js'))`

A1 - 2 SQL and NoSQL Injection (fixes can be found in allocations.html and allocations-dao.js)

- `1'; return 1 == '1`
- `';while(true){};'`

A1 - 3 Log Injection

- `curl http://localhost:4000/login -X POST --data 'userName=vyva%0aError: alex moldovan failed $1,000,000 transaction&password=Admin_123&_csrf='`

A2:2017 Pérdida de Autenticación

Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son implementadas incorrectamente, permitiendo a los atacantes comprometer usuarios y contraseñas, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios (temporal o permanentemente).

Ejemplos en NodeGoat

A2 - 1 Session Management

<http://nodegoat.herokuapp.com/tutorial/a2>

A3:2017 Exposición de datos sensibles

Muchas aplicaciones web y APIs no protegen adecuadamente datos sensibles, tales como información financiera, de salud o Información Personalmente Identificable (PII). Los atacantes pueden robar o modificar estos datos protegidos inadecuadamente para llevar a cabo fraudes con tarjetas de crédito, robos de identidad u otros delitos. Los datos sensibles requieren métodos de protección adicionales, como el cifrado en almacenamiento y tránsito.

Ejemplos en NodeGoat

A6 - Sensitive Data Exposure

<http://nodegoat.herokuapp.com/tutorial/a6>

A7:2017 Secuencia de Comandos en Sitios Cruzados (XSS)

Los XSS ocurren cuando una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada; o actualiza una página web existente con datos suministrados por el usuario utilizando una API que ejecuta JavaScript en el navegador. Permiten ejecutar comandos en el navegador de la víctima y el atacante puede secuestrar una sesión, modificar (defacement) los sitios web, o redireccionar al usuario hacia un sitio malicioso.