

Oscar Manuel Ruiz Hurtado

10

Fundamentos y Técnicas de Seguridad para Aplicaciones

Evaluación de fin de cursado

1. Gestión de Dependencias como Problema de Seguridad

Analiza por qué la gestión de dependencias de software (ej. en NPM, PyPI, apt) se ha convertido en un vector crítico de ataques a la cadena de suministro. Propón dos prácticas, una técnica y una organizacional, para concientizar o reducir este riesgo.

2. Antipatrones de seguridad

¿Cuáles de los siguientes son *antipatrones de seguridad* reconocidos? Indica cuáles son, e indica qué correspondería hacer si encuentras código con este *antipatrón* en un sistema que estés trabajando.

- a) "Security by Obscurity" (Seguridad por Oscuridad).
- b) Almacenar contraseñas en texto plano en la base de datos.
- c) Implementar el principio de mínimo privilegio para los servicios.
- d) Realizar validación de entrada únicamente del lado del cliente.

3. Desbordamientos

Si bien los desbordamientos más famosos (y por buenas razones!) son los desbordamientos de pila (*stack overflows*), la cantidad de amenazas que pueden caracterizarse de esta manera es mucho mayor. Compara la explotación de un *desbordamiento de pila* (*stack overflow*) con un *desbordamiento de enteros* (*integer overflow*).

4. Enumeración de debilidades, enumeración de vulnerabilidades

Dos de los índices más importantes para la comprensión y el seguimiento de la seguridad en aplicaciones son el **CWE** (*Common Weakness Enumeration*) y el **CVE** (*Common Vulnerabilities Enumeration*). ¿Cuál es la relación entre ambos? ¿Existe algún *camino* que lleve a algún *problema* que descubramos en determinado código a *transitarse* de clasificarse como CWE a clasificarse como CVE (o a la inversa)?

5. Ataques de inyección

Elige un ataque de inyección específico (SQLi, des-serialización, SSRF, etc.) y propón una estrategia de "defensa en profundidad", ubicando controles de seguridad distintos en diferentes capas (ej. aplicación, red, SO) para mitigarlo.

6. Tipos de inyección

Relaciona cada concepto (*tipo de inyección*) con su descripción correspondiente.

Tipo de inyección

Descripción

- | | |
|---|--|
| <input checked="" type="checkbox"/> a) Inyección de comandos | a) La aplicación des-serializa datos no confiables, instanciando objetos o invocando métodos arbitrarios |
| <input checked="" type="checkbox"/> b) Inyección de objetos | b) Datos no validados se envían a un intérprete de comandos del sistema operativo |
| <input checked="" type="checkbox"/> c) Server-Side Request Forgery (SSRF) | c) Entradas de usuario se incorporan directamente en una consulta de base de datos, alterando su lógica |
| <input checked="" type="checkbox"/> d) Inyección SQL | d) La aplicación es engañada para realizar consultas a un destino interno o restringido |

Oscar Manuel Roiz Hurtado

① La gestión de dependencias de SW en si se ha convertido en un vector crítico en la seguridad porque hoy en día el código está expuesto en su total a lo max un 10% por novedades y en 90% por otros que usamos el código para nuestro fin y el de otros cosa usando el de otros y así en cadena. Por ello si una dependencia a cualquier nivel de la cadena se vulnera, este afectaría los niveles de cima. Las 2 prácticas recomendables serían

- Usar manejadores de dependencias como Maven para estar actualizado y parchando con esta vulnerabilidad en las depend.

~~② Proy. Concentración constante (técnica)~~

- ~~es importante, así~~
- Concentrizar de no impactar nada más porque si, a la vez suficiente. Mejor para una función muy sencilla, un paquete entero que podría dar pautas tóxicas y ser/ciencias de que lo menos. Es más (organización)

② a) El usuario asume que nadie sabe cómo funciona su código por defecto y por ello no hace robusto sus mecanismos de seguridad. Mecanismo: Seguir buenas prácticas con pautas fijas en cada nivel de nuestra implementación

b) El usuario deja sin cifrar contraseñas y si la base se roba o se explota alguna vulnerabilidad de lectura el atacante puede volar contraseñas de usuarios.

Mecanismo: Hashear las contraseñas y guardarlos hashes; a la hora del login comprobar hashes solamente

c) Escribir un antipattern: consiste en dar al usuario el mínimo número de permisos que le permite realizar su trabajo

d) Asume que lo que envía el cliente está bien sin tu validar las entradas por lo que da problemas de explotación. Mecanismo: Siempre validar entradas, con su codificación y en un middleware para centralizar validaciones.

③ El desbordamiento de pila ~~excede el límite de~~ la pila que almacena las variables locales y por ende los returns comprende el flujo del código corrupto y así se explota esto que se da saltos de flujo inesperados. Un Atacante puede usar esto para saltar y ejecutar código formido a otro flujo que explota el programa con los returns, el de entradas surge cuando se le manda un número que no cabe en la memoria asignada ~~más que en la memoria en el tipo de datos~~.
dado otro número y esto podría dar pie a otra lógica no esperada, por ejemplo un atacante que le daiga una edad muy alta a una Promoción y el overflow de entero podría pasarlo a una edad temporal y acceder a un documento de mico por ejemplo algo.

0x0 → Un int overflow también puede llevar a ejecución de código arbitrario aunque requiere mucha más "talento" → CVE-2025-7985

④ Los CVE buscan trazar un histórico de los casos en dónde se han encontrado vulnerabilidades, los encargados de Seguridad trazan estos índices para ver cuales vulnerabilidades y cómo mitigarlos o si sus proyectos incluyen dependencias con otras vulnerabilidades y parcharlos (actualizándolas) o manipulándolas en su código si aún no existe parche. Este índice permite que las vulnerabilidades se conozcan rápido en la comunidad y se trabaje de forma colaborativa para mitigar los riesgos. Por otro lado CVE busca notar prácticas de programación que son candidatas a sufrir vulnerabilidades y los programadores las sustituyan con alternativas más seguras. Por lo tanto un CVE podría formalizarse para ser un CVE que funcionaría como recordatorio para que los programadores no repitan la historia.

Oscar Mandi Ruiz Hurlado

⑤ Inyección de objetos - Defensa en profundidad

Nivel aplicación:

- Autenticación del usuario (un usuario autenticado no intenta atacar?)
- Implementación del mínimo privilegio ✓
- Usar des-serializadores seguros (comprobados) que revisan codificación y validación de datos esperados. ✓
- Centralizar en un módulo wäre todo la validación y no en todo el código ✓

Nivel Red: usar protocolos seguros como SSH o HTTPS
~~creo que estas~~ Usar sesiones con tiempos cortos de vida
medios no tendrían mayor efecto ✓

Nivel SO: manejo de acceso por privilegios
por atributos a datos fijos del programa
(con mayor granularidad)