

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 *transitar* 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

a) Inyección de comandos

a) La aplicación des-serIALIZA datos no confiables, instanciando objetos o invocando métodos arbitrarios

b) Inyección de objetos

b) Datos no validados se envían a un intérprete de comandos del sistema operativo

c) Server-Side Request Forgery (SSRF)

c) Entradas de usuario se incorporan directamente en una consulta de base de datos, alterando su lógica

d) Inyección SQL

d) La aplicación es engañada para realizar consultas a un destino interno o restringido

# Oscar Manuel Ruiz 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 escrito en su total o lo max un 10% por nosotros y "un 90%" por otros que usamos el código para nuestro fin y el de otros está usando el de otros y así en cadena. Por ello si una dependencia a cualquier nivel de la cadena, se vulnera, este afecta a los niveles por encima. Las 2 prácticas recomendables serán:
- Usar manejadores de dependencias como maven para estar actualizando y parchando con esto vulnerabilidad en las depend. constantemente (técnica)
  - Concietizar de no impactar nada mas porque si, a lo mejor para una función muy sencilla, un paquete entero que podría dar puertas traseras y ser concientes de que lo menos es más (organizacional)

- ② a) El <sup>desarrollador?</sup> usuario asume que nadie sabe cómo funciona su código por dentro y por ello no hace robusto sus mecanismos de seguridad. Mecanismo: Seguir buenas prácticas con patrones seguros en cada nivel de nuestra implementación.
- b) El usuario deja sin entrar contraseña y si la base se roba o se explota alguna vulnerabilidad de lectura el atacante puede robar contraseñas de usuarios. Mecanismo: Hashear las contraseñas y guardarlos hashes; a la hora del login comparar hashes solamente.
- c) Este NBOS un antipatrón: consiste en darle al usuario el mínimo número de permisos que le permite realizar sus trabajos.
- d) Asume que lo que envía el cliente está bien sin su validación. Mecanismo: Siempre validar entradas, con su codificación y en un middleware para centralizar validaciones.

varios de las vulnerabilidades y ataques que vimos vienen de lograr subir sofisticadamente a estos repositorios típicamente confiables. ¿Cómo podemos prevenir eso?

Org - la concietización es importante, pero una vez suficiente.

③ El desbordamiento de pila ~~se debe~~ <sup>se debe</sup> a la pila que almacena las variables locales y por ende los returns reemplazando el flujo del código corriendo y si se explota esto puede dar saltos de flujo inesperados. Un atacante puede usar esto para saltar y escalar permisos o a otro flujo que explota el programa con los returns, el de enteros surge cuando se le manda un número que no cabe en la memoria asignada dando otro número y esto podría dar pie a otra lógica no esperada, por ejemplo un atacante que le da una edad muy alta a una promoción y de el overflow de entero podría pasarlo a una edad temprana y acceder a un descuento de más por tener algo.

más que en la memoria en el tipo de datos.  
Por ejemplo, un entero 16 bits desborda el signo  $65535 + 1 = 0 (2^{16})$

Ojo → un int. overflow también puede llevar a ejecución de código arbitrario aunque requiere mucho más "talento" → CVE-2025-7985

④ Los CVE buscan trazar un historico de los casos en donde se han encontrado vulnerabilidades, los encargados de Seguridad trackean estos índices para ver nuevas vulnerabilidades y cómo mitigarlos o si sus proyectos incluyen dependencias con estas vulnerabilidades y parchearlas (actualizándolas) o o manejarlas 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 los sustituyen con alternativas más seguras. Por lo tanto un CVE podría formalizarse para ser un CVE que funcionaría como recomendación para que los programadores no repitan la historia!

Oscar Manuel Ruiz Huilaco

### ③ 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 (corroborados)
- que revisen codificación y validación de datos ✓  
esperados.
- Centralizar en un módulo ~~where~~ toda la validación  
y no en todo el código ✓

Nivel Red: ~~Usar protocolos seguros como SSH o HTTPS~~  
*creo que está* Usar sesiones con tiempos ~~cerbos de vida~~ ✓  
*medidos not tendrían mayor efecto*

Nivel SO: Mensaje de acceso por privilegios  
por atributos q datos fuera del programa  
(con mayor granularidad) ✓