Temario de Estudio: Desarrollo Seguro de Software

Temario de Estudio: Desarrollo Seguro de Software

1. Introducción al Desarrollo Seguro

Importancia del Desarrollo Seguro

- El desarrollo seguro es fundamental para prevenir vulnerabilidades y evitar que atacantes exploten fallos en las aplicaciones.
- La seguridad debe integrarse desde el inicio del desarrollo y no tratarse como un complemento adicional.

Impacto de las Vulnerabilidades Comunes

- **Cross-Site Scripting (XSS):** Inyección de scripts maliciosos en sitios web para robar información o manipular la interfaz.
- **SQL Injection:** Inserción de código SQL malicioso para acceder o modificar datos en bases de datos.
- **Cross-Site Request Forgery (CSRF):** Engaño a usuarios autenticados para que realicen acciones sin su consentimiento.
- **Deserialización Insegura:** Ejecución de código malicioso al procesar datos serializados sin validación adecuada.
- **Falta de cifrado en tránsito y en reposo:** Exposición de datos sensibles al ser interceptados en la comunicación o al almacenarse sin encriptación.

Riesgos Derivados de la Falta de Seguridad en Aplicaciones

- Pérdida de datos y filtraciones de información.
- Pérdida de confianza por parte de los usuarios.
- Sanciones legales y cumplimiento de normativas como GDPR, ISO 27001 y PCI DSS.

2. Requisitos de Seguridad

Identificación de Elementos Críticos a Proteger

- **Entradas del usuario:** Validar y sanitizar todos los datos provenientes de formularios, URLs, headers, etc.
- **Datos sensibles:** Información personal (PII), credenciales de usuarios, tokens de acceso, claves API.
- **Variables de entorno:** Uso adecuado de variables de entorno para evitar exponer credenciales en el código.

Identificación de Vectores y/o Puntos de Ataque

- **Entrada de datos:** Manipulación de formularios y peticiones.
- **Interfaces de programación (APIs):** Exposición de endpoints sin autenticación adecuada.
- **Servicios en la nube:** Configuración incorrecta de permisos y accesos.
- **Bibliotecas y dependencias:** Uso de paquetes desactualizados o con vulnerabilidades.

Buenas Prácticas Iniciales

- **Políticas de contraseñas seguras:** Uso de contraseñas fuertes y almacenamiento seguro con hashing (bcrypt, Argon2).
- **Estándares y arquitectura segura:** Aplicar principios de arquitectura segura como Zero Trust y microservicios con controles de acceso.

3. Diseño Seguro

Principios de Seguridad en el Diseño

- **Principio del Mínimo Privilegio (PoLP):** Asignar los permisos estrictamente necesarios a usuarios y sistemas.
- **Defensa en profundidad:** Uso de múltiples capas de seguridad (firewalls, autenticación multifactor, auditoría de logs).
- **Seguridad por diseño:** Considerar la seguridad como un elemento esencial desde la fase de diseño.

Separación de Datos Sensibles del Código

- **Uso de vaults seguros (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault).**
- **No almacenar contraseñas o claves en repositorios públicos.**

Control y Revisión de Dependencias

- Auditoría frecuente de bibliotecas de terceros con herramientas como `npm audit`, `yarn audit` y `OWASP Dependency-Check`.

Importancia de la Modularidad y la Separación de Responsabilidades

- Aplicar principios de arquitectura modular y desacoplamiento para minimizar el impacto de vulnerabilidades en un solo módulo.

4. Desarrollo Seguro

Validación y Sanitización de Entradas y Salidas

- Uso de validadores como **Joi**, **express-validator** en Node.js.
- Rechazo de caracteres especiales para evitar inyecciones de código.

Protección de Datos Sensibles

- **Cifrado:** AES-256 para datos en reposo.
- **Hashing:** Bcrypt, Argon2 para contraseñas.
- **HMAC:** Para integridad de datos en APIs.

Uso Adecuado de Tokens

- **ACCESS_TOKEN:** Tokens de corta duración para autenticación.
- **REFRESH_TOKEN:** Tokens de larga duración almacenados de manera segura.

Malas Prácticas en el Desarrollo de Software

- **Exposición de claves API en commits.**
- **Uso de configuraciones por defecto inseguras.**

Configuración de Políticas de Seguridad

- **CORS:** Restricción de orígenes permitidos.
- **Roles y permisos:** Implementación de RBAC (Role-Based Access Control).

5. Gestión de Dependencias

Uso de Herramientas para Auditoría

- `npm audit`, `yarn audit` para detectar vulnerabilidades en paquetes.

Importancia de Mantener las Dependencias Actualizadas

- Uso de versiones activamente mantenidas con soporte de seguridad.

6. Pruebas de Seguridad

Implementación de Análisis de Seguridad

- **Pruebas de penetración (Pentesting).**
- **Revisión de código estático con SonarQube, Snyk, Checkmarx.**

Revisión y Auditoría Constante del Código

- Implementar revisiones de seguridad en ciclos de desarrollo (DevSecOps).

7. Ataques Comunes y sus Contramedidas

Descripción y Mitigación de Ataques

- **XSS:** Escapar entradas y usar CSP (Content Security Policy).
- **SQL Injection:** Uso de consultas parametrizadas y ORM seguro.
- **DDoS:** Protección con WAF (Web Application Firewall) y rate-limiting.
- **Clickjacking:** Uso de `X-Frame-Options: DENY` en encabezados HTTP.
- **Fuerza Bruta:** Implementación de bloqueos tras intentos fallidos.

Medidas de Protección Específicas

- **Firewalls de Aplicación Web (WAF).**
- **Autenticación multifactor (MFA).**

- **Rotación periódica de credenciales y claves API.**

8. Buenas Prácticas en Seguridad

Principios Claves

- **"Nunca confíes en el usuario":** Toda entrada debe ser validada y sanitizada.
- **Separación entre datos sensibles y lógica de negocio.**
- **Revisión y actualización continua de dependencias.**
- **Auditoría constante del código y la infraestructura.**
- **"Si sirve no lo toques... al menos que te paguen demasiado bien":** Mantener la estabilidad sin comprometer la seguridad.
