

Seguretat Informatica (SI)

Tema 5. Seguridad en las aplicaciones

Davide Careglio

Fuentes: A. Rodriguez, “Tema 6: Seguretat a les aplicacions”, Computer Security, 2018

Temario

- ▶ Tema 1. Introducción
- ▶ Tema 2. Criptografía
- ▶ Tema 3. Infraestructura PKI
- ▶ Tema 4. Seguridad en la red
- ▶ **Tema 5. Seguridad en las aplicaciones**
- ▶ Tema 6. Seguridad en los sistemas operativos
- ▶ Tema 7. Análisis forense

Tema 5. Seguridad en las aplicaciones

Índice

- ▶ Introducción
- ▶ Errores más comunes (según OWASP)
- ▶ Gestión de vulnerabilidades
- ▶ Auditorias

Tema 5. Seguridad en las aplicaciones

Índice

- ▶ **Introducción**
- ▶ Errores más comunes (según OWASP)
- ▶ Gestión de vulnerabilidades
- ▶ Auditorias

5.1 – Introducción

- ▶ **No existe un sistema 100% seguro**
 - ▶ Vulnerabilidades teóricas
 - ▶ Vulnerabilidades reales (exploits)
- ▶ **La debilidad de un sistema permite a un atacante violar la**
 - ▶ Confidencialidad
 - ▶ Integridad
 - ▶ Disponibilidad

5.1 – Introducción

Causas de software inseguro

- ▶ Vulnerabilidades
- ▶ Estructura organizativa, procesos de desarrollo, tecnología
- ▶ Desarrolladores
- ▶ Requisitos legales
- ▶ Incremento de la conectividad y complejidad

5.1 – Introducción

Soluciones

- ▶ Asegurar que un programa de seguridad sea efectivo y rentable usando SDLC
 - ▶ Systems Development Life Cycle
- ▶ Entender el alcance de la seguridad que requiere un proyecto
- ▶ Probar pronto y a menudo
- ▶ Pensar creativamente
- ▶ Utilizar las herramientas adecuadas
- ▶ Utilizar código fuente cuando esté disponible
- ▶ Documentar los resultados de las pruebas

5.1 – Introducción

Soluciones

- ▶ Asegurar que un programa de seguridad sea efectivo y rentable usando SDLC



5.1 – Introducción

¿por qué programación segura?

- ▶ Las aplicaciones web son la forma más fácil de atacar un sistema
- ▶ La seguridad NO puede ser un problema ajeno al desarrollador web
- ▶ Hay que ser consciente de los problemas y de las posibles soluciones
- ▶ Hay que suponer que los usuarios NO usarán una aplicación únicamente como la ha diseñado el desarrollador
- ▶ No hay que confiar en todo lo que envían los usuarios aunque se considere que usan aplicaciones seguras

5.1 – Introducción

¿por qué programación segura?

- ▶ Las...
- siste
- ▶ La s
- des
- ▶ Hay
- solu
- ▶ Hay
- única
- ▶ No
- se c



TRUST NO ONE

n

bles

cación

aunque

Tema 5. Seguridad en las aplicaciones

Índice

- ▶ Introducción
- ▶ Errores más comunes (según OWASP)
- ▶ Gestión de vulnerabilidades
- ▶ Auditorias

5.2 – Errores más comunes

OWASP

- ▶ Open Web Application Security Project
- ▶ <https://owasp.org>
- ▶ Fundación sin ánimo de lucro
- ▶ Participación abierta y gratuita
- ▶ Libre de presiones corporativas
- ▶ Creada con la finalidad de determinar y combatir las causas que hacen que el software sea inseguro
- ▶ Pretende establecer métodos de trabajo seguro a la hora de desarrollar aplicaciones web

5.2 – Errores más comunes

OWASP Top10 – 2017

► Define la lista de los 10 errores más comunes

[https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_\(en\).pdf.pdf](https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_(en).pdf.pdf)

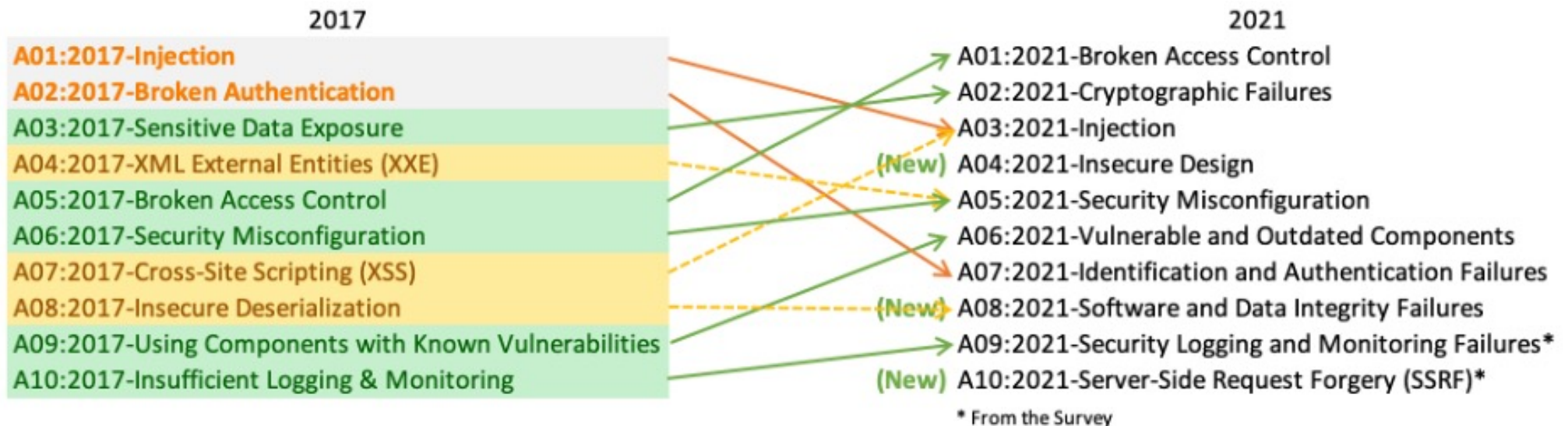
OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

5.2 – Errores más comunes

OWASP Top10 - 2021

- Define la lista de los 10 errores más comunes

<https://owasp.org/Top10/>



5.2 – Errores más comunes

Top10

- ▶ Broken Access Control
- ▶ Cryptographic Failures
- ▶ Injection
- ▶ Insecure design
- ▶ Security Misconfiguration
- ▶ Vulnerable and Outdated Components
- ▶ Identification and Authentication Failures
- ▶ Software and Data Integrity Failures
- ▶ Security Logging and Monitoring Failures
- ▶ Server-Side Request Forgery

5.2 – Errores más comunes

Top10

- ▶ Broken Access Control
- ▶ Cryptographic Failures
- ▶ Injection
- ▶ Insecure design
- ▶ Security Misconfiguration
- ▶ Vulnerable and Outdated Components
- ▶ Identification and Authentication Failures
- ▶ Software and Data Integrity Failures
- ▶ Security Logging and Monitoring Failures
- ▶ Server-Side Request Forgery

5.2.1 – Broken Access Control

- ▶ Pasa de la posición 5 (2017) a la 1 (2021)
- ▶ El control de acceso debe hacer cumplir la política de seguridad de modo que los usuarios no puedan actuar fuera de sus permisos previstos
 - ▶ Las restricciones sobre lo que los usuarios pueden hacer a menudo no se aplican de manera adecuada
 - ▶ Un atacante puede explotar estos defectos para acceder a información no autorizada, puede modificar o destruir datos o realizar una función fuera de los límites previstos

5.2.1 – Broken Access Control

Vulnerabilidades más comunes

- ▶ Violación del principio de least privilege o deny by default
 - ▶ El acceso está disponible para cualquier persona y no según los roles, capacidades o usuarios específicos
- ▶ Eludir las comprobaciones de control de acceso
 - ▶ Pej., modificando la URL (i.e., parameter tampering), el estado de la aplicación, la página HTML o las solicitudes a través de un API
- ▶ Permitir ver o editar la cuenta de otra persona
 - ▶ Usando su identificador único (no cifrado o no modificado en la sesión)
- ▶ Acceso a través de una API donde faltan controles de acceso
- ▶ Elevación de privilegios
 - ▶ Actuar como usuario sin iniciar sesión o actuar como administrador cuando inicie sesión como usuario
- ▶ Configuración incorrecta de Cross-Origin Resource Sharing (CORS)
 - ▶ Permite el acceso a la API desde orígenes no autorizados o no confiables

5.2.1 – Broken Access Control

Ejemplo 1

- ▶ Una aplicación usa datos sin verificar en un llamada SQL para acceder a la información de una cuenta

```
pstmt.setString(1, request.getParameter("acct"));  
ResultSet results = pstmt.executeQuery( );
```

- ▶ Un atacante puede modificar el parámetro **acct** en el navegador para acceder a la información de cualquier cuenta

```
https://example.com/app/accountInfo?acct=notmyacct
```

5.2.1 – Broken Access Control

Ejemplo 2

- ▶ Un atacante poder ir probado URL con acceso a páginas de administración

```
https://example.com/app/getappInfo  
https://example.com/app/admin_getappInfo
```

- ▶ Un usuario no autenticado NO puede acceder a cualquiera de las páginas, es un posible punto de ataque
- ▶ Una persona que no es administrador NO puede acceder a la página de administración, es otro posible punto de ataque

5.2.1 – Broken Access Control

Prevención

- ▶ Aplicar políticas de deny by default a todos los recursos que no son públicos
- ▶ Implementar mecanismos de control de acceso y reusarlo durante todo el periodo de acceso a la aplicación
 - ▶ Minimizar el uso de CORS
- ▶ Aplicar controles sobre las propiedades de cada dato, de manera que un usuario no pueda manipular datos de otros
- ▶ Aplicar rate-limit a los accesos/uso de APIs
- ▶ Deshabilitar la opción de listar el directorio del servidor web y asegurar que los metadatos y los archivos de backup no estén presentes
- ▶ Controlar los logs de fallos de acceso repetidos y hacer saltar alarmas
- ▶ En servidores stateful, los identificadores de sesiones deben invalidarse una vez terminada la sesión
- ▶ En servidores stateless, los tokens pueden ser sujetos a vulnerabilidades
 - ▶ Reducir la duración de los tokens y usarlos solo en caso específico y seguros

5.2.2 – Cryptographic Failures

- ▶ Antes llamado “Sensitive Data Exposure”
- ▶ Pasa de la posición 3 (2017) a la 2 (2021)
- ▶ Cuando se protege de forma incorrecta información confidencial en un sistema o en tránsito entre sistemas, como los datos financieros y/o médicos, tarjetas bancarias, secretos de negocio, etc.
- ▶ Los atacantes pueden robar o modificar dichos datos si son protegidos débilmente para realizar fraudes, robo de identidad u otros delitos

5.2.2 – Cryptographic Failures

Vulnerabilidades más comunes

- ▶ Envío de tráfico en texto claro
 - ▶ Por ejemplo, tráfico interno entre servidores, entre sistemas de back-end
- ▶ Uso de algoritmos de cifrados obsoletos o débiles, códigos antiguos o usados por defecto cuando fallan los principales
 - ▶ P. e., al pedir un parámetro usado solo en algoritmos obsoletos
- ▶ Uso de claves por defecto, débiles, reusadas o sin modificar periódicamente
- ▶ Cifrado no aplicado
 - ▶ P. e., no se usan todos los parámetros de seguridad en la cabecera HTTP
- ▶ No se ha validado correctamente el certificado o la cadena de confianza
- ▶ Uso de funciones de hash obsoletas como MDI o SHA-1
- ▶ Uso de métodos de padding obsoletos como PKCS#1 v1.5

5.2.2 – Cryptographic Failures

Ejemplos

▶ Ejemplo 1

- ▶ Una aplicación cifra números de tarjetas de crédito en una base de datos usando cifrado automático de la base de datos
- ▶ Sin embargo, estos datos se descifran automáticamente cuando se accede a ellos
- ▶ Esto permite que, al hacer un ataque de inyección SQL, el atacante recupere los números de tarjetas de crédito sin cifrar

▶ Ejemplo 2

- ▶ Una aplicación web no usa TLS para todas las páginas o usa un cifrado débil
- ▶ Un atacante puede monitorear la conexión, pasar la conexión de HTTPS a HTTP y robar la cookie de sesión del usuario
- ▶ Con la cookie, el atacante puede secuestrar la sesión autenticada del usuario y acceder/modificar sus datos privados

5.2.2 – Cryptographic Failures

Prevención

- ▶ Clasificar los datos que procesa, almacena o transmite una aplicación según su grado de confidencialidad de acuerdo a la legislación actual
- ▶ Descartar toda aquella información confidencial que no es necesaria guardar (eliminarla incluido de las caches)
- ▶ Cifrar toda la información en transito usando protocolos seguros como TLS
- ▶ Usar los algoritmos, protocolos y gestores de claves estandarizados y más actualizados posibles
- ▶ Evitar mostrar paginas de error por defecto ya que estas pueden contener información confidencial sobre los sistemas
- ▶ Almacenar las contraseñas de forma segura
 - ▶ Veremos luego que significa

5.2.2 – Cryptographic Failures

Prevención

- ▶ Evitar mostrar paginas de error por defecto ya que estas pueden contener información confidencial sobre los sistemas

Estado HTTP 500 - Error de Sintaxis en sentencia SQL "SELECT * FROM COMENTARIS WHERE AUTOR="[*]"

type Informe de Excepción

mensaje Error de Sintaxis en sentencia SQL "SELECT * FROM COMENTARIS WHERE AUTOR="[*]"

descripción El servidor encontró un error interno que hizo que no pudiera rellenar este requerimiento.

excepción

```
java.lang.RuntimeException: Error de Sintaxis en sentencia SQL "SELECT * FROM COMENTARIS WHERE AUTOR='[*]'"
Syntax error in SQL statement "SELECT * FROM COMENTARIS WHERE AUTOR='[*]'"; SQL statement:
SELECT * FROM COMENTARIS WHERE AUTOR='' [42000-188]
    edu.upc.escert.curs.repositori.vulnerable.RepositoriComentaris.getComentarisFromSQL(RepositoriComentaris.java:62)
    edu.upc.escert.curs.repositori.vulnerable.RepositoriComentaris.getComentarisPerAutor(RepositoriComentaris.java:78)
    edu.upc.escert.curs.LlistaComentarisServlet.doGet(LlistaComentarisServlet.java:31)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
```

nota La traza completa de la causa de este error se encuentra en los archivos de diario de Apache Tomcat/7.0.39.

Apache Tomcat/7.0.39

5.2.2 – Cryptographic Failures

Prevención

- ▶ **Guardar las contraseñas de forma segura**
 - ▶ No guardarlas nunca en claro
 - ▶ Generar un hash para cada contraseña de la base de datos (BD)
 - ▶ Comparar el hash de la contraseña introducida con el de la BD
- ▶ **Puede no ser suficiente**
 - ▶ Raibow tables: <https://crackstation.net/>
Tener el hash de una contraseña simple es como tener la contraseña
- ▶ **Hay que usar el Salt**
 - ▶ <https://crackstation.net/hashing-security.htm>
 - ▶ Se concatena un valor aleatorio (llamado Salt) a la contraseña antes de generar el hash de la BD
 - ▶ Se guarda este Salt como valor asociado al usuario
 - ▶ Y se concatena con la contraseña introducida para generar el hash y compararla con el de la BD

5.2.3 – Injection

- ▶ Pasa de la posición 1 (2017) a la 3 (2021)
- ▶ Se incluye el Cross-Site Scripting (XSS) en esta categoría
- ▶ Se produce cuando se pasa información no validada a un interprete como parte de un comando o consulta
- ▶ Y este, pensando que son legítimos, los ejecuta o permite al acceso a datos sin la autorización adecuada
- ▶ Los ataques de inyección más comunes son a SQL, NoSQL, comandos del SO, LDAP, etc.

5.2.3 – Injection

Ejemplo

- ▶ En el caso SQL

```
String query = "SELECT * FROM accounts WHERE custID='" +  
request.getParameter("id") + "'";
```

- ▶ El atacante puede modificar el valor del parámetro id en su navegador y enviar una opción que siempre es cierta

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

- ▶ Esto cambia el significado de la consulta para devolver todos los registros de la tabla account
- ▶ Los ataques más peligrosos podrían modificar o eliminar datos, o incluso invocar procedimientos almacenados

5.2.3 – Injection

Cross-Site Scripting (XSS)

- ▶ XSS permite a los atacantes ejecutar scripts en el navegador de la víctima que pueden robar sesiones de usuario, manipular sitios web o redirigir al usuario a sitios maliciosos
- ▶ Ocurren cuando una aplicación
 - ▶ incluye datos no confiables en un enlace sin una validación o escape adecuados
 - ▶ o actualiza una página web existente con datos proporcionados por el usuario utilizando una API que permite ejecutar HTML o JavaScript

5.2.3 – Injection

Cross-Site Scripting (XSS)

- ▶ XSS permite a los atacantes ejecutar scripts en el

nav
usu
ma

▶ Oc

▶ il

▶ e

▶ c

▶ f

▶ j



es de
a sitios

ción o

rcionados
HTML o

5.2.3 – Injection

Prevención

- ▶ La opción preferida es usar una API segura, que evita el uso del intérprete por completo o proporciona una interfaz parametrizada y validada
- ▶ Utilizar la validación de entrada positiva o *while-list* del lado del servidor
- ▶ Utilizar LIMIT y otros controles SQL sobre el número y tipos de consultas para evitar la divulgación masiva de registros en caso de inyección SQL
- ▶ Usar la sintaxis de escape específica para evitar el uso de caracteres especiales que validen una consulta
- ▶ Filtrar según la semántica de la aplicación, por ejemplo
 - ▶ **L'Hospitalet** en un nombre de ciudad valido
 - ▶ **importante** puede ser un comentario valido si la aplicación permite un campo con texto enriquecido

5.2.4 – Insecure design

- ▶ Nueva categoría en 2021
- ▶ El diseño inseguro es una categoría amplia que representa diferentes debilidades, expresadas como diseño de control no presente o ineficaz
- ▶ Diseño e implementación son aspectos diferentes
 - ▶ Un diseño seguro puede seguir teniendo una implementación insegura
 - ▶ Un diseño inseguro no se puede “salvar” con una implementación segura
- ▶ Muchas veces un diseño es inseguro por no haber creado perfiles de riesgo y, por lo tanto, no se ha incluido una solución desde cero

5.2.4 – Insecure design

Ejemplo 1

- ▶ Un sistema ofrece la posibilidad de recuperar las credenciales de un usuario a través de preguntas y respuestas, por ejemplo

Indica el nombre de tu primera mascota

- ▶ No se puede confiar en las preguntas y respuestas como evidencia de identidad, ya que más de una persona puede conocer las respuestas
- ▶ Este tipo de recuperación de credenciales están prohibidas actualmente
- ▶ Dicho código debe eliminarse y reemplazarse por un diseño más seguro

5.2.4 – Insecure design

Ejemplo 2

- ▶ Un sitio web de venta de tarjetas de video de alta gama no tiene un sistema de protección contra bots de compra masiva
- ▶ Los atacantes compran continuamente todo el stock en el momento en el cual salen a la venta que luego lo revenden en sitios de subastas
- ▶ Los compradores solo pueden acceder a precios más elevados
- ▶ La web obtiene mala reputación
- ▶ Se necesita un sistema anti-bot que deniegue determinadas compras según reglas específicas
 - ▶ P.e., compra masiva a los pocos segundos de estar disponible

5.2.4 – Insecure design

Prevención

- ▶ Integrar verificaciones de plausibilidad en cada nivel de la aplicación (desde el front-end hasta el back-end)
- ▶ Establecer y utilizar un SDLC con profesionales en Application Security para ayudar en el diseño
- ▶ Utilizar un modelo de amenazas para la autenticación crítica, el control de acceso, la lógica y el intercambio de claves
- ▶ Preparar tests de validación para cada bloque y para su integración y comprobar que son resistentes al modelo de amenazas
 - ▶ Preparar tests tanto para casos de uso legítimo como de uso indebido
- ▶ Segregar y aislar los grupo de usuarios que comparten un acceso común
- ▶ Limitar el consumo de recursos por parte del usuario o servicio

5.2.5 – Security Misconfiguration

- ▶ Pasa de la posición 6 (2017) a la 5 (2021)
- ▶ Se incluye ahora la categoría del 2017 llamada XML External Entities (XXE)
- ▶ Todo lo que concierne una configuración de seguridad débil, mal implementada, o directamente faltante

5.2.5 – Security Misconfiguration

Vulnerabilidades más comunes

- ▶ Falta del nivel de seguridad adecuado en cualquier parte de las aplicaciones o permisos configurados incorrectamente
- ▶ Hay funciones habilitadas o instaladas que son innecesarias
 - ▶ P. e., puertos, servicios, páginas, cuentas o privilegios innecesarios
- ▶ Cuentas, claves o contraseñas predeterminadas aún habilitadas
- ▶ El software está desactualizado o es vulnerable
- ▶ El servidor no envía cabeceras o directivas de seguridad, o no están configurados con valores seguros
- ▶ Las últimas funciones de seguridad en los sistemas actualizados están deshabilitadas o no configuradas de forma segura
- ▶ El gestor de errores revela a los usuarios información innecesaria

5.2.5 – Security Misconfiguration

Ejemplos

▶ Ejemplo 1

- ▶ El servidor viene con aplicaciones demo que no se han eliminado del servidor en producción
- ▶ Estas aplicaciones demo tienen fallos de seguridad conocidas que los atacantes utilizan para comprometer el servidor
- ▶ Si por ejemplo, una de estas aplicaciones es la consola de administración y no se han modificado las cuentas predeterminadas, el atacante puede iniciar sesión con las contraseñas predeterminadas y tomar el control del servidor

▶ Ejemplo 2

- ▶ La lista de directorios no está deshabilitada en el servidor
- ▶ Un atacante puede descubrir este acceso y encontrar y descargar las clases Java compiladas
- ▶ Descompilando y aplicando ingeniería inversa, puede descubrir el código fuente y analizarlo para encontrar una vulnerabilidad

5.2.5 – Security Misconfiguration

XML External Entities (XXE)

- ▶ Consiste en conseguir que una aplicación con un parser XML reciba o ejecute un código malicioso
 - ▶ Un XML puede contener un Document Type Definition (DTD) que, entre otras cosas, permite la definición de entidades XML referenciadas a través de un URI
 - ▶ De esta forma, al ejecutar un parser XML mal configurados se puede colar un URI a documentos u objetos confidenciales o forzar conexiones externas maliciosa bypassando las reglas del firewall
- ▶ De esta forma se puede conseguir
 - ▶ Archivos internos
 - ▶ Escaneo de puertos internos
 - ▶ Ejecución remota de código
 - ▶ Ataques de denegación de servicio

5.2.5 – Security Misconfiguration

XML External Entities (XXE) - ejemplos

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<foo>&xxe;</foo>
```

Se consigue acceder al fichero de las contraseña en sistema Linux/Unix

```
<!ENTITY xxe SYSTEM "https:192.168.1.1/private">]>
```

Se consigue información sobre una dirección interna privada

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Ataque DoS
(billion laughs)

5.2.5 – Security Misconfiguration

Prevención

- ▶ Configurar una tarea para revisar los niveles de seguridad y actualizar aplicaciones y instalar parches
- ▶ Revisar los permisos de almacenamiento en la nube
- ▶ Mantener en producción una plataforma mínima sin atributos, componentes, documentación ni ejemplos innecesarios, eliminando funciones y frameworks no utilizados
- ▶ Definir y configurar los entornos de desarrollo, control de calidad y producción de forma idéntica pero con diferentes credenciales en cada entorno
- ▶ Definir un proceso automatizado para verificar la efectividad de las configuraciones y ajustes en todos los entornos
 - ▶ Deshabilitar la opción DTD para evitar XXE
 - ▶ Validar los ficheros XML de entrada
 - ▶ Actualizar los parsers

5.2.6 – Vulnerable and Outdated Comp.

- ▶ Antes conocido como “Using Components with Known Vulnerabilities”
- ▶ Pasa de la posición 9 (2017) a la 6 (2021)
- ▶ Todo lo que concierne el uso interno o externo de aplicaciones, protocolos, librerías, SO, etc. que contienen vulnerabilidades y que no están actualizados

5.2.6 – Vulnerable and Outdated Comp.

Problemas más comunes

- ▶ Cuando no se conocen las versiones de todos los componentes que se utilizan (tanto del lado del cliente como del lado del servidor)
 - ▶ Esto incluye los componentes que se usan directamente, así como las dependencias
- ▶ Cuando el software es vulnerable, incompatible o no actualizado
 - ▶ SO, servidor web, el administrador de bases de datos, las API y las librerías
- ▶ Cuando no se busca vulnerabilidades con regularidad
- ▶ Cuando no se asegura las configuraciones de los componentes
 - ▶ Relacionado con 5.2.5
- ▶ Cuando no se prueba la compatibilidad con las librerías modificadas, actualizadas o parcheadas
- ▶ Cuando no se repara o actualiza la plataforma de base, los frameworks y las dependencias de manera oportuna

5.2.6 – Vulnerable and Outdated Comp. Ejemplo

► Vulnerabilidad en Zoom desde la versión 5.03 a 5.12 para Windows

Posted by [Mitja Kolsek](#) on [July 09, 2020](#)

Remote Code Execution Vulnerability in Zoom Client for Windows (0day)

by Mitja Kolsek, the Opatch Team

Earlier this week a security researcher shared a remote code execution "0day" vulnerability in Zoom Client for Windows with our team. The vulnerability allows a remote attacker to execute arbitrary code on victim's computer where Zoom Client for Windows (any currently supported version) is installed by getting the user to perform some typical action such as opening a document file. No security warning is shown to the user in the course of attack.

The researcher (who wants to keep their identity private) stated that they did not report the vulnerability to Zoom either directly or through a broker, but would not object to us reporting it to Zoom.

[Update 7/13/2020: Zoom only took one (!) day to issue a new version of Client for Windows that fixes this vulnerability, which is remarkable. We have reviewed their fix and can confirm that it efficiently resolves the vulnerability. With an official vendor fix available to all users, we made our micropatches for this issue PRO-only according to our [guidelines](#). Meanwhile, after issuing micropatches for this issue targeted at Zoom Client for Windows versions 5.0.3 to 5.1.2, we noticed a lot of our users being on all of these versions despite Zoom's highly persistent update mechanism. We had expected most users to be on version 5.1.2, but this indicates many users may still be on even older Zoom Client versions. We therefore ported our micropatch to the remaining supported versions of Zoom Client: 5.0.0, 5.0.1, and 5.0.2. We're now covering all vulnerable supported clients.]

5.2.6 – Vulnerable and Outdated Comp.

Prevención

- ▶ Eliminar las dependencias no utilizadas, las funciones, los componentes, los archivos y la documentación innecesarios
- ▶ Realizar un inventario continuo de las versiones de los componentes (p. e., frameworks, librerías) y sus dependencias utilizando herramientas como OWASP Dependency Check, retire.js, etc.
- ▶ Supervisar continuamente fuentes como la Common Vulnerability and Exposures (CVE) y la National Vulnerability Database (NVD) para las vulnerabilidades en los componentes
- ▶ Obtener componentes de fuentes oficiales a través de enlaces seguros, usando paquetes firmados para reducir la posibilidad de incluir un componente malicioso modificado
 - ▶ Relacionado con 5.2.8 - Software and Data Integrity Failures
- ▶ Monitorear las librerías y los componentes que no se mantienen o no crean parches de seguridad para versiones antiguas

5.2.7 – Identification and Auth. Failures

- ▶ Pasa de la posición 2 (2017) a la 7 (2021)
- ▶ Antes conocido como “Broken authentication”
- ▶ Todo lo que concierne los problemas relacionados con el proceso de login, gestión de sesiones y credenciales

5.2.7 – Identification and Auth. Failures

Vulnerabilidades más comunes

- ▶ Usar contraseñas débiles como Solarwinds | 23
- ▶ Permitir contraseñas por defecto como admin, password, | 234
- ▶ Permitir ataques automáticos y de fuerza bruta
- ▶ Usar métodos de recupero de contraseña poco seguros
 - ▶ Relacionado con 5.2.4 – Insecure design
- ▶ Autenticación simple sin control multipaso/multifactor
- ▶ Mantener la base de datos de las contraseñas con poca seguridad
 - ▶ Relacionado con 5.2.2 – Cryptographic Failures
- ▶ Mantenimiento de las sesiones con ID únicos o expuestos
- ▶ Exponer el ID de las sesiones en el URL

5.2.7 – Identification and Auth. Failures

Prevención

- ▶ **Asegurar que las conexiones HTTP usen un canal seguro y cifrado**
 - ▶ Usar HTTPs y TLS cuando se envían usuarios y contraseñas y durante toda la conexión para que nadie vea las cookies de sesión
- ▶ **Usar Cookie “Secure”**
 - ▶ Para que las cookies se envíen solo si se está usando HTTPs
- ▶ **Usar HTTPOnly como atributo**
 - ▶ Para que las cookies no se vean en javascript

5.2.7 – Identification and Auth. Failures

Prevención

- ▶ Implementar un verificador de contraseñas débiles
 - ▶ Top 10,000 worst passwords
<https://github.com/danielmiessler/SecLists/tree/master/Passwords>
 - ▶ Usar las recomendaciones sobre complejidad, longitud y duración de las contraseñas
<https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret>
- ▶ Implementar técnicas de autenticación multifactor para evitar
 - ▶ Ataques masivos y automáticos
 - ▶ Robo de credenciales (credential stuffing)
 - ▶ Fuerza bruta

5.2.7 – Identification and Auth. Failures

Prevención

- ▶ **Evitar el problema del “Session Fixation”**
 - ▶ Se produce cuando se entra en una sesión y esta se mantiene con el mismo ID del login y no tiene un temporizador
 - ▶ Un atacante podría volver a entrar pasado un tiempo ya que la sesión sigue abierta
 - ▶ Un usuario podría pasar la URL a otro usuario y este acceder con las credenciales del primero usando el ID del login
- ▶ **Para evitar este problema**
 - ▶ Usar un administrador de sesión seguro del lado del servidor que genere una nueva ID de sesión aleatoria con alta entropía después del inicio de sesión
 - ▶ Los ID de sesión no deben estar en la URL, deben almacenarse de forma segura e invalidarse después de cerrar sesión o pasado un tiempo de inactividad

5.2.8 – Software and Data Integrity Failures

- ▶ Nueva categoría en 2021
- ▶ Se incluye el “Insecure deserialization” en esta categoría
- ▶ Todo lo que concierne los problemas de seguridad relacionados con hacer suposiciones cuando se realizan actualizaciones de software, de datos críticos y de canales de CI/CD sin verificar la integridad

5.2.8 – Software and Data Integrity Failures

Problemas más comunes

- ▶ Cuando una aplicación se basa en complementos, librerías o módulos obtenidos de fuentes, repositorios y/o CDN que no son de confianza
- ▶ Cuando un canal de CI/CD es poco seguro y un atacante puede obtener acceso no autorizado, introducir código malicioso o comprometer el sistema
- ▶ Cuando se realizan actualizaciones automáticas y esas se descargan sin una verificación de integridad suficiente
 - ▶ Los atacantes podrían cargar sus propias actualizaciones para distribuir las y ejecutarlas en todas las instalaciones
 - ▶ Recordar el caso del software Orion de SolarWinds (Tema I)

5.2.8 – Software and Data Integrity Failures

Ejemplo

- ▶ Actualización maliciosa de SolarWinds
- ▶ La empresa que desarrolla el software Orion tenía procesos de construcción y actualización seguros y con control de integridad
- ▶ Aún así, estos pudieron ser subvertidos (recordar la contraseña)
- ▶ Durante varios meses, la empresa distribuyó una actualización con código malicioso a más de 18.000 clientes
- ▶ Se trata de una de las infracciones de esta naturaleza más trascendentes y significativas de la historia

5.2.8 – Software and Data Integrity Failures

Insecure deserialization

- ▶ Se produce cuando, al procesar y deserializar unos datos que ha enviado un atacante, se consigue introducir un objeto que introduce o modifica el comportamiento de una aplicación
- ▶ Este objeto puede ser
 - ▶ El contenido de una Remote Procedure Call
 - ▶ Una cookie
 - ▶ Tokens de una API
 - ▶ Una llamada a un web service
- ▶ Nota:
 - ▶ Serializar: convertir un objeto en un formato (ejemplos: binario, XML/JSON) que se puede guardar o enviar a un periférico (stdout, red, etc.)
 - ▶ Deserializar: proceso inverso, se recibe un dato serializado y se reconvierte en el objeto original

5.2.8 – Software and Data Integrity Failures

Insecure deserialization - ejemplo

- Un forum usa serialización de objetos PHP para guardar “super-cookies” que contienen el ID de los usuarios, sus roles, sus contraseñas y otros estados

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";  
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

- Un atacante puede cambiar el objeto serializado para darse privilegios de administrador

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";  
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```


5.2.8 – Software and Data Integrity Failures

Prevención

- ▶ Utilizar firmas digitales o mecanismos similares para verificar que el software o los datos provengan de la fuente de confianza
- ▶ Asegurar que las librerías y dependencias usen repositorios de confianza
- ▶ Asegurar que haya un proceso de revisión de los cambios de código y configuración para minimizar la posibilidad de que se introduzcan códigos maliciosos o se abran brechas de seguridad
- ▶ Asegurar que el canal de CI/CD tenga segregación y control de acceso adecuados para garantizar la integridad del código
- ▶ No aceptar objetos serializados de fuentes no confiables o usar verificación de integridad o firma digital para detectar alteraciones

Tema 5. Seguridad en las aplicaciones

Índice

- ▶ Introducción
- ▶ Errores más comunes (según OWASP)
- ▶ **Gestión de vulnerabilidades**
- ▶ Auditorias

5.3 – Gestión de vulnerabilidades

Common Platform Enumeration



- ▶ Se necesitan procesos de inventariado de software altamente confiables y automatizables
- ▶ CPE proporciona:
 - ▶ Un formato estándar legible por máquina para codificar nombres de productos y plataformas.
 - ▶ Un conjunto de procedimientos para comparar nombres
 - ▶ Un lenguaje para construir "declaraciones de aplicabilidad" que combinen nombres de CPE con operadores lógicos simples

5.3 – Gestión de vulnerabilidades

Common Platform Enumeration – ejemplos

 **cpe:2.3:o:microsoft:windows_10:1703:*:*:*:*:*:***

[Hide Matching CPE\(s\)](#) ▲

- *cpe:2.3:o:microsoft:windows_10:1703:*:*:*:*:**
- *cpe:2.3:o:microsoft:windows_10:1703:*:*:*:*:x64:*
- *cpe:2.3:o:microsoft:windows_10:1703:*:*:*:*:x86:*

 **cpe:2.3:o:microsoft:windows_10:1709:*:*:*:*:*:***

[Hide Matching CPE\(s\)](#) ▲

- *cpe:2.3:o:microsoft:windows_10:1709:*:*:*:*:**
- *cpe:2.3:o:microsoft:windows_10:1709:*:*:*:*:arm64:*
- *cpe:2.3:o:microsoft:windows_10:1709:*:*:*:*:x64:*
- *cpe:2.3:o:microsoft:windows_10:1709:*:*:*:*:x86:*

5.3 – Gestión de vulnerabilidades

Common Vulnerabilities and Exposures

- ▶ Identificadores que se usan para indicar una vulnerabilidad
- ▶ Definido y mantenido por MITRE Corporation
 - ▶ <https://cve.mitre.org>

CVE-ID	
CVE-2019-1359	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
A remote code execution vulnerability exists when the Windows Jet Database Engine improperly handles objects in memory, aka 'Jet Database Engine Remote Code Execution Vulnerability'. This CVE ID is unique from CVE-2019-1358.	
References	
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.	
<ul style="list-style-type: none">• MISC:https://portal.msrmicrosoft.com/en-US/security-guidance/advisory/CVE-2019-1359	
Assigning CNA	
Microsoft Corporation	
Date Record Created	
20181126	Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.
Phase (Legacy)	
Assigned (20181126)	

5.3 – Gestión de vulnerabilidades

Common Vulnerability Score System (CVSS)

- ▶ Definido a partir del año 2004
- ▶ Se trata de un sistema de puntaje diseñado para proveer un método abierto y estándar que permite estimar el impacto derivado de vulnerabilidades
- ▶ CVSS se encuentra bajo la custodia de Forum of Incident Response and Security Teams (FIRST), pero se trata de un estándar completamente abierto
- ▶ <https://www.first.org/cvss>
- ▶ Actualmente versión 3.1



5.3 – Gestión de vulnerabilidades

Common Vulnerability Score System (CVSS) - ejemplo

► CVE-2019-1359

CVSS v3.1 Severity and Metrics:

Base Score: 7.8 HIGH

Vector: AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

Impact Score: 5.9

Exploitability Score: 1.8

Attack Vector (AV): Local

Attack Complexity (AC): Low

Privileges Required (PR): None

User Interaction (UI): Required

Scope (S): Unchanged

Confidentiality (C): High

Integrity (I): High

Availability (A): High

5.3 – Gestión de vulnerabilidades

National Vulnerability Database

- ▶ El NIST dispone de una base de datos con todas las vulnerabilidades que han aparecido desde 2002
 - ▶ <https://nvd.nist.gov>
- ▶ Permite la consulta directa o bien descargar todos los datos en formato XML
- ▶ Se actualiza diariamente.
 - ▶ Agrega los CVSS a posteriori, entre 8 y 48 horas después normalmente



5.3 – Gestión de vulnerabilidades

National Vulnerability Database - ejemplo

🚩 CVE-2019-1359 Detail


Current Description

A remote code execution vulnerability exists when the Windows Jet Database Engine improperly handles objects in memory, aka 'Jet Database Engine Remote Code Execution Vulnerability'. This CVE ID is unique from CVE-2019-1358.

[+ View Analysis Description](#)

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 **NIST:** NVD **Base Score:** 7.8 HIGH **Vector:** CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

QUICK INFO

CVE Dictionary Entry:

[CVE-2019-1359](#)

NVD Published Date:

10/10/2019

NVD Last Modified:

08/24/2020

Source:

Microsoft Corporation

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites. Please address comments about this page to nvd@nist.gov.

Hyperlink	Resource
https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-1359	Patch Vendor Advisory

5.3 – Gestión de vulnerabilidades

Aplicación

- ▶ Para poder usar toda esta información sobre vulnerabilidades a nuestra organización, es necesario pero primero tener un sistema de inventariado adecuado que permita saber
 - ▶ Cuales son los activos de la organización
 - ▶ Donde se encuentran
 - ▶ Que funciones desarrollan
 - ▶ En que estado se encuentran

5.3 – Gestión de vulnerabilidades

0-days

- ▶ Las vulnerabilidades 0-days son aquellas que se han descubierto sin que el fabricante del producto lo sepa
- ▶ Estas vulnerabilidades se pueden poner a disposición de la comunidad y permiten la creación de exploits
- ▶ Durante el periodo de tiempo disponible entre este descubrimiento y que el fabricante solucione el problema, hay claramente una criticidad alta

5.4 Seguridad en las aplicaciones

Índice

- ▶ Introducción
- ▶ Errores más comunes (según OWASP)
- ▶ Gestión de vulnerabilidades
- ▶ Auditorias

5.4 – Auditorias

- ▶ Definición dada por la Real Academia de la Lengua Española:
 - ▶ La auditoría consiste en la revisión sistemática de una actividad o de una situación para evaluar el cumplimiento de las reglas o criterios objetivos a los que las mismas deben ser sometidas
- ▶ En nuestro caso consiste en revisar la seguridad y la integridad de aplicaciones o servidores y evaluar las posibles vulnerabilidades encontradas

5.4 – Auditorias

Técnicas

- ▶ Inspecciones y revisiones manuales
- ▶ Modelado de amenazas
- ▶ Revisión de código
- ▶ Pruebas de intrusión

5.4 – Auditorias

Técnicas

- ▶ Inspecciones y revisiones manuales
- ▶ Modelado de amenazas
- ▶ **Revisión de código**
- ▶ Pruebas de intrusión

5.4.1 – Revisión de código

- ▶ Es buena idea realizar análisis del código que escribimos
 - ▶ Mejor si lo hace otra persona
 - ▶ O un proceso ...
- ▶ Se puede analizar el código de manera
 - ▶ Estática (caja blanca)
 - ▶ Dinámica (en ejecución)

5.4.1 – Revisión de código

Análisis estático

- ▶ **Permite comprobar:**
 - ▶ Que se siguen buenas prácticas de programación
 - ▶ Vulnerabilidades típicas
- ▶ Se le pueden escapar errores de diseño propios del lenguaje o la aplicación en sí misma
- ▶ **Según la fase del SDLC**
 - ▶ Codificación
 - ▶ Pruebas



5.4.1 – Revisión de código

Análisis estático

- ▶ **Algunas herramientas**
 - ▶ PMD
 - ▶ SonarQube
 - ▶ Coverity
 - ▶ FxCop: per Microsoft .NET
 - ▶ FinBugs: busca patrones de error en Java
 - ▶ FlawFinder: per c/c++
 - ▶ RIPS: per PHP
 - ▶ Bandit: per Python

5.4.1 – Revisión de código

Análisis dinámico

- ▶ Permite comprobar el comportamiento de la aplicación en un entorno de ejecución real
- ▶ Pueden aparecer errores no detectados en el análisis estático
- ▶ Herramientas:
 - ▶ El debugger de siempre
 - ▶ Application Verifier: herramienta Microsoft
 - ▶ ProcessMonitor

5.4 – Auditorias

Técnicas

- ▶ Inspecciones y revisiones manuales
- ▶ Modelado de amenazas
- ▶ Revisión de código
- ▶ Pruebas de intrusión

5.4.2 – Pruebas de intrusión

▶ Ventajas

- ▶ Puede ser rápido y barato
- ▶ Requieren un conocimiento relativamente menor que una revisión del código fuente
- ▶ Comprueban el código que realmente está expuesto

▶ Inconvenientes

- ▶ Demasiado tarde desde el punto de vista del SDLC
- ▶ Pruebas sólo los impactos frontales

5.4.2 – Pruebas de intrusión

¿Son legales?

- ▶ Según el artículo 197 bis del código penal en referencia al acceso no autorizado a sistemas informáticos:
 - ▶ 1. El que por cualquier medio o procedimiento, vulnerando las medidas de seguridad establecidas para impedirlo, y **sin estar debidamente autorizado**, acceda o facilite a otro el acceso al conjunto o una parte de un sistema de información o se mantenga en él en contra de la voluntad de quien tenga el legítimo derecho a excluirlo, será castigado con pena de prisión de seis meses a dos años
 - ▶ 2. El que mediante la utilización de artificios o instrumentos técnicos, y **sin estar debidamente autorizado**, intercepte transmisiones no públicas de datos informáticos que se produzcan desde, hacia o dentro de un sistema de información, incluidas las emisiones electromagnéticas de los mismos, será castigado con una pena de prisión de tres meses a dos años o multa de tres a doce meses

5.4.2 – Pruebas de intrusión

Pasos

- ▶ Information gathering
- ▶ Escaneo
- ▶ Análisis manual o automática
- ▶ Explotación/obtención de evidencias

5.4.2 – Pruebas de intrusión

Pasos

- ▶ **Information gathering**
- ▶ Escaneo
- ▶ Análisis manual o automática
- ▶ Explotación/obtención de evidencias

5.4.2 – Pruebas de intrusión

Information gathering

- ▶ A la hora de hacer una auditoría es importante obtener el máximo de información posible sobre los servidores o los servicios auditados
- ▶ No se trata sólo de informarnos, sino también de buscar “conocimientos indebidos” como pueden ser la estructura interna de la red, archivos que deberían estar ocultos ...
- ▶ Es decir, queremos encontrar información que no deberíamos (o deberían) de encontrar

5.4.2 – Pruebas de intrusión

Information gathering - ejemplos

- ▶ **Mediante peticiones DNS a los servidores es posible obtener mucha información**
 - ▶ Número de servidores de correo / DNS (redundancia)
 - ▶ Servicios externos (caching, correo)
 - ▶ Subdominios
 - ▶ Servidores virtualizados

- ▶ **Hay varias herramientas que permiten realizar estas peticiones**
 - ▶ nslookup
 - ▶ dig
 - ▶ fierce

5.4.2 – Pruebas de intrusión

Information gathering – Google hacking

- ▶ Es una técnica que utiliza el buscador de google con parámetros específicos que permiten filtrar por URL o ciertos strings que nos pueden permitir descubrir archivos internos de servidores que hayan sido indexados
- ▶ Existencia de operadoras especiales
 - ▶ operador: término
 - ▶ Sin espacios y operador en minúsculas
- ▶ El término puede ser
 - ▶ Una palabra
 - ▶ Una frase (entre comillas)
- ▶ Existen operadores que deben utilizarse solos
 - ▶ Empiezan con la palabra reservada ALL
- ▶ Si se produce un error de sintaxis se entiende como un término más a buscar

5.4.2 – Pruebas de intrusión

Information gathering – Google hacking

- ▶ **Herramientas de automatización de google hacking**
 - ▶ FoundStone Sitedigger
 - ▶ Apollo
 - ▶ Athena
 - ▶ Wikto
- ▶ **Google Hacking Database (GHDB)**
 - ▶ Colección de técnicas de Google hacking
 - ▶ <https://www.exploit-db.com/google-hacking-database>

5.4.2 – Pruebas de intrusión

Information gathering – Metadatos

- ▶ Muchos documentos contienen información extra
- ▶ Los metadatos pueden proporcionar información
 - ▶ Usuarios
 - ▶ Localización
 - ▶ IPs
 - ▶ Software
- ▶ Herramientas
 - ▶ stat
 - ▶ pdftinfo
 - ▶ file
 - ▶ <https://metashieldclean-up.elevenpaths.com>

5.4.2 – Pruebas de intrusión

Information gathering – Metadatos

- ▶ **Fingerprinting Organization with Collected Archives (FOCA)**
 - ▶ Herramienta que busca metadatos e información escondida en los documentos
 - ▶ Combina y automatiza muchas técnicas
 - ▶ Permite extraer y organizar información de los archivos encontrados



5.4.2 – Pruebas de intrusión

Information gathering – Deep Web & Pastebin

- ▶ Siempre es buena idea buscar apariciones de la entidad que estamos auditando a la Deep Web (con cuidado) y en Pastebin, ya que muchos grupos de “moralidad dudosa” suelen dejar allí la información que obtienen

5.4.2 – Pruebas de intrusión

Pasos

- ▶ Information gathering
- ▶ **Escaneo**
- ▶ Análisis manual o automática
- ▶ Explotación/obtención de evidencias

5.4.2 – Pruebas de intrusión

Escaneo

- ▶ Una de las primeras cosas que necesitamos saber es qué servicios de la entidad que auditamos se encuentran "al descubierto", es decir, accesibles desde fuera
- ▶ Queremos realizar un escaneo para detectar cuáles IPs y puertos están abiertos y cuáles sirven ofrecen a través suyo
- ▶ Hay varias herramientas útiles para automatizar esta tarea

5.4.2 – Pruebas de intrusión

Escaneo - nmap

- ▶ Herramienta ejecutable por línea de comandos de Linux muy potente para realizar escáneo de red, puertos, SO, servicios, versiones, etc.

```
nmap -v -A scanme.nmap.org
```

- ▶ Escaneo detallado con detección de SO, versiones, scripts y traceroutes.

```
nmap -F scanme.nmap.org
```

- ▶ Escaneo rápido que sólo detecta los puertos abiertos o filtrados.



5.4.2 – Pruebas de intrusión

Escaneo - nmap

► Ejemplo

```
nmap -O scanme.nmap.org
```

```
Starting Nmap 6.47 ( http://nmap.org ) at 2015-11-09 16:31 CET
```

```
Nmap scan report for scanme.nmap.org (45.33.32.156)
```

```
Host is up (0.17s latency).
```

```
Not shown: 990 closed ports
```

PORT	STATE	SERVICE
22/tcp	open	ssh
25/tcp	filtered	smtp
53/tcp	filtered	domain
80/tcp	open	http
135/tcp	filtered	msrpc
139/tcp	filtered	netbios-ssn
445/tcp	filtered	microsoft-ds
593/tcp	filtered	http-rpc-epmap
9929/tcp	open	nping-echo
31337/tcp	open	Elite

```
Aggressive OS guesses: Linux 3.11 - 3.13 (95%), Linux 3.2 - 3.8 (95%), IPFire firewall 2.11  
(Linux 2.6.32) (94%), Linux 2.6.32 (94%), Linux 3.1 - 3.2 (94%), DD-WRT v24-sp1 (Linux 2.4)  
(93%), Linux 2.6.32 - 2.6.39 (92%), IGEL UD3 thin client (Linux 2.6) (92%), Linksys WRV200  
wireless broadband router (92%), DD-WRT v24-sp2 (Linux 2.4.36) (92%)
```

```
No exact OS matches for host (test conditions non-ideal).
```

```
Network Distance: 18 hops
```

```
OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
```

```
Nmap done: 1 IP address (1 host up) scanned in 14.42 seconds
```

5.4.2 – Pruebas de intrusión

Pasos

- ▶ Information gathering
- ▶ Escaneo
- ▶ **Análisis manual o automática**
- ▶ Explotación/obtención de evidencias

5.4.2 – Pruebas de intrusión

Análisis manual o automática

- ▶ Una vez descubierta toda la información posible, se pasa a analizar y descubrir las vulnerabilidades
- ▶ Esta tarea puede ser
 - ▶ Automática: usando herramientas disponibles que realicen tests de penetración usando los ataques más comunes
 - ▶ Manual: un experto con conocimientos actualizados (documentarse y actualizarse son factores importante!), intuición y creatividad es necesario para hacer pruebas más exhaustivas y completas

5.4.2 – Pruebas de intrusión

Análisis automática

- ▶ Hay muchas herramientas disponibles para hacer análisis automatizada
- ▶ Ejemplos
 - ▶ Wfuzz: herramienta muy conocida y altamente configurable que permite atacar servicios mediante fuzzing
 - ▶ sqlmap: para realizar tests de penetración a bases de datos
 - ▶ OWASP ZAP: permite el análisis de aplicaciones web, haciendo de proxy entre el navegador y el servicio a auditar
 - ▶ OpenVAS: herramienta que permite escanear una infraestructura, detectar los servicios que ofrece y automatizarla explotación de las vulnerabilidades

5.4.2 – Pruebas de intrusión

Pasos

- ▶ Information gathering
- ▶ Escaneo
- ▶ Análisis manual o automática
- ▶ **Explotación/obtención de evidencias**

5.4.2 – Pruebas de intrusión

Explotación

- ▶ Una vez detectadas las (posibles) vulnerabilidades se pueden explotar
- ▶ Hay que tener cuidado por varias razones
 - ▶ Hay cumplir el acuerdo con el responsable del sistema
 - ▶ Hay que evitar “tumbar” involuntariamente el sistema

5.4.2 – Pruebas de intrusión

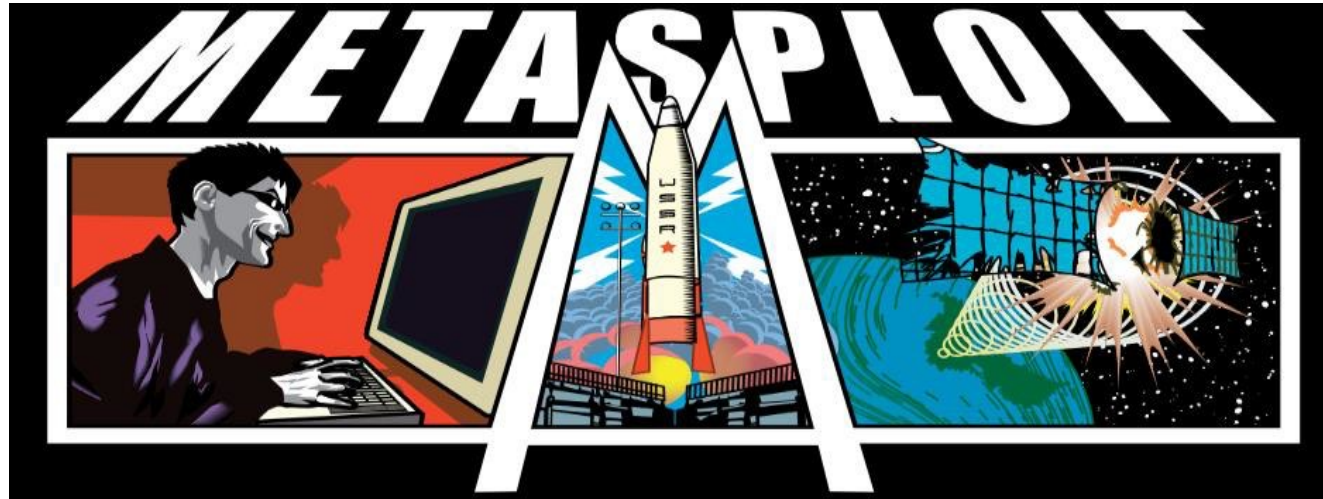
Explotación

- ▶ El primer paso para explotar una vulnerabilidad es mirar si ya existe algún exploit conocido
- ▶ Algunas son relativamente sencillas de comprobar:
 - ▶ Un archivo que debería estar oculto es accesible.
 - ▶ Podemos inyectar SQL en un campo concreto.
- ▶ Hay otros que necesitan hacer uso de exploit más complejas y también tenemos que mirar si ya existen:
 - ▶ Podemos buscarlos a exploit-db
 - ▶ <https://www.exploit-db.com/>



5.4.2 – Pruebas de intrusión

Explotación - Metasploit

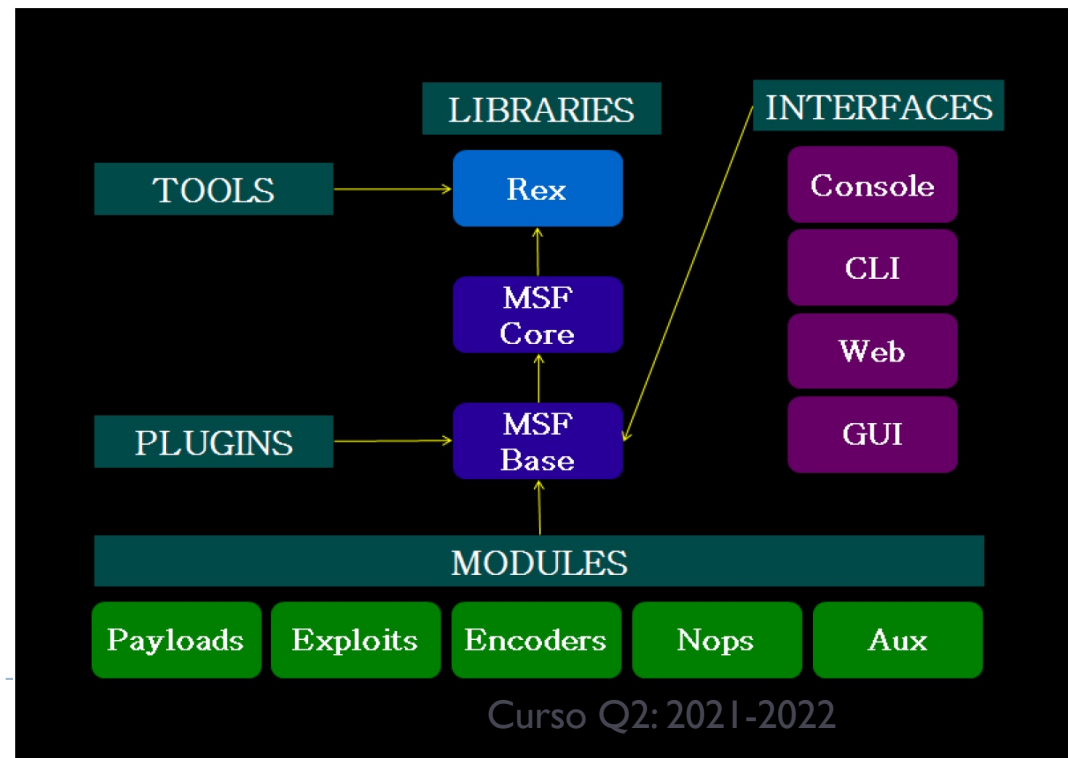


- ▶ Una herramienta que permite
 - ▶ Desarrollar exploit
 - ▶ Ejecutar exploit en maquinas locales o remotas
 - ▶ Escanear maquinas buscando vulnerabilidades
 - ▶ Recoger información sobre vulnerabilidades

5.4.2 – Pruebas de intrusión

Explotación - Metasploit

- ▶ El framework también contiene un conjunto de binarios del tipo *msf* que permiten, entre otros:
 - ▶ Invocar una línea de encomienda para interactuar con Metasploit
 - ▶ Ejecutar una interfaz gráfica para interactuar con Metasploit.
 - ▶ Ofuscación de los payloads mediante encoders
 - ▶ Generación de payloads
 - ▶ Análisis de binarios



Seguretat Informatica (SI)

Tema 5. Seguridad en las aplicaciones

Davide Careglio