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

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

- 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
Soluciones

- Asegurar que un programa de seguridad sea efectivo y rentable usando SDLC
- 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
- La
- Hay
- Hay
- No

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 animo 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

- Define la lista de los 10 errores más comunes

  https://owasp.org/www-project-top-ten/
  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
Top10

- Injection
- Broken Autentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Secutiry Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging&Monitoring

5.2 – Errores más comunes
Top10

- Injection
- Broken Autentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Secutiry Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging&Monitoring

5.2.1 – Injection

- Se produce cuando se pasa información no validada a un interprete como parte de un comando o consulta
- Y este, pensando que los ha escrito el programador, los ejecuta o permite al acceso a datos sin la autorización adecuada

5.2.1 – 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 por ejemplo

```
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

5.2.1 – Injection
Prevención

- Mantener los datos separados de los comandos y consultas
- La opción preferida es usar una API segura, que evita el uso del intérprete por completo o proporciona una interfaz parametrizada
- Utilizar la validación de entrada positiva o *while-list* del lado del servidor
- Utilizar LIMIT y otros controles SQL en las consultas para evitar la divulgación masiva de registros en caso de inyección SQL
- Para cualquier consulta dinámica, usar la sintaxis de escape específica para evitar caracteres especiales

5.2.2 – Broken autentication

- Todo lo que concierne los problemas relacionados con el proceso de login, gestión de sesiones y credenciales
  - Usar contraseñas débiles
  - Permitir ataques automáticos y de fuerza bruta
  - Permitir contraseñas por defecto como admin, password, 1234
  - Usar métodos de recupero de contraseña poco seguros
  - Mantener la base de datos de las contraseñas con poca seguridad
  - Mantenimiento de las sesiones con ID únicos o expuestos (Session Fixation)
  - Autenticación simple sin control multipaso/multifactor

5.2.2 – Broken autentication
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

5.2.2 – Broken autentication

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

5.2.2 – Broken autentication
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

5.2.3 – Sensitive Data Exposure

- Cuando se protege de forma incorrecta información confidencial en un sistema o en transito entre sistemas, como los datos financieros y médicos
- Los atacantes pueden robar o modificar dichos datos débilmente protegidos para realizar fraudes con tarjetas de crédito, robo de identidad u otros delitos
- Los datos confidenciales pueden verse comprometidos sin protección adicional, como el cifrado en reposo o en tránsito

5.2.3 – Sensitive Data Exposure
Ejemplos

- Una aplicación cifra números de tarjetas de crédito en una base de datos usando encriptación automática de la base de datos. Sin embargo, estos datos se descifran automáticamente cuando se recuperan, lo que permite que un ataque de inyección SQL recupere números de tarjetas de crédito en texto claro
- La base de datos de contraseñas utiliza hashes simples para almacenar las contraseñas de todos. Un fallo en la carga de archivos permite a un atacante recuperar la base de datos de contraseñas. Las contraseñas se pueden descubrir usando una tabla rainbow de hashes precalculados.

5.2.3 – Sensitive Data Exposure
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
- Evitar mostrar paginas de error por defecto ya que estas pueden contener información confidencial sobre los sistemas

5.2.3 – Sensitive Data Exposure
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

5.2.3 – Sensitive Data Exposure
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

5.2.4 – 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

5.2.4 – 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

```
<
<!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)

5.2.4 – XML External Entities (XXE)
Prevención

- Deshabilitar la opción DTD
- Validar los ficheros XML de entrada
- Actualizar los parsers
- Hacer referencias a las buenas practicas sobre como deshabilitar el procesado de XXE
  https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html
- Implementar *white-list*, filtrado y saneamiento (*sanitization*) a todos los inputs recibidos del lado servidor
- Utilizar formatos simples como JSON
- Utilizar herramientas automáticas para detectar XXE

5.2.5 – Broken Access Control

- Las restricciones sobre lo que los usuarios autenticados pueden hacer a menudo no se aplican de manera adecuada
- Los atacantes pueden explotar estos defectos para acceder a la funcionalidad y / o datos no autorizados, como acceder a las cuentas de otros usuarios, ver archivos confidenciales, modificar los datos de otros usuarios, cambiar los derechos de acceso, etc.

5.2.5 – Broken Access Control
Ejemplo

- 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

5.2.5 – Broken Access Control
Prevención

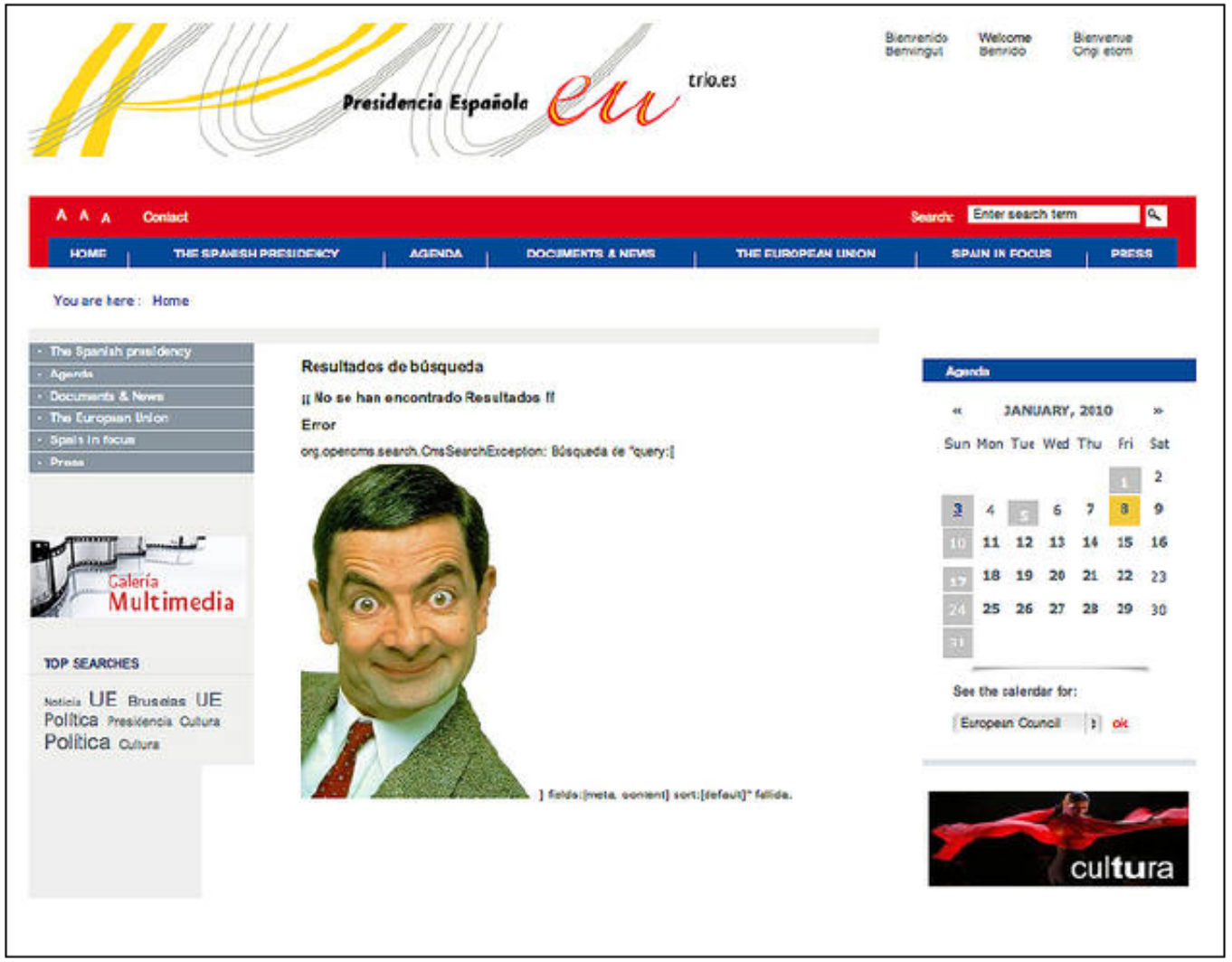
- Aplicar políticas de denegación por defecto a todos los recursos que no son públicos
- Implementar mecanismos de control de acceso para validar los datos enviados/recibidos
- Aplicar controles sobre las propiedades de cada dato, de manera que un usuario no pueda manipular datos de otros
- Aplicar rate-limit a los accesos/APIs

5.2.6 – 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

5.2.6 – Cross-Site Scripting (XSS)

- XSS permite a los atacantes ejecutar scripts en el nav … es de usu … a sitios ma …
- Oc …
  - i … ción o e …
  - c … rcionados … HTML o J …



5.2.6 – Cross-Site Scripting (XSS)
Prevención

- Mantra: filter on input, escape on output

  http://lukeplant.me.uk/blog/posts/why-escape-on-input-is-a-bad-idea/
- Filtrar según la semántica de la aplicación, por ejemplo
  - **L'Hospitalet** en un nombre de ciudad valido
  - **In&Out** es un titulo de una película valido
  - **SELECT * FROM PROVA WHERE ID='1'** puede ser una respuesta valida a un cuestionario sobre SQL
  - **importante** puede ser un comentario valido si la aplicación permite un campo con texto enriquecido
- Usando herramientas/librerías que escapan automáticamente de XSS por diseño
  - System.Web.Security.AntiXss.AntiXssEncoder Class para .NET 4.5 to 4.8
  - ValidateRequest function para ASP.NET 2.0

5.2.7 – 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

5.2.7 – 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";}
```

5.2.7 – Insecure deserialization
Prevención

- La única técnica segura es no aceptar objetos serializados de fuentes no confiables o usar medios de serialización que solo permitan tipos de datos primitivos
- Si no es posible, entonces:
  - Implementar validación de integridad
  - Forzar el uso de restricciones a la hora de deserializar un objeto antes de ejecutarlo
  - Aislar y ejecutar código que deserializa objetos en entornos con privilegios de acceso limitados
  - Monitorizar las conexiones de entrada y salida de los servidores que deserializan objetos

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:/a:microsoft:internet_explorer:11:beta`
  - `cpe:/a:microsoft:internet_explorer:10`
  - `cpe:/a:microsoft:internet_explorer:9`
  - `cpe:/a:microsoft:internet_explorer:8`

5.3 – Gestión de vulnerabilidades
Common Vulnerabilities and Exposures

- Identificadores que se usan para indicar una vulnerabilidad
- Ejemplos
  - CVE – CVE-2014-0268
  - CPE
    - cpe:/a:microsoft:internet_explorer:11:
    - cpe:/a:microsoft:internet_explorer:10
    - cpe:/a:microsoft:internet_explorer:9
    - cpe:/a:microsoft:internet_explorer:8
  - Descripción
    - Existe una vulnerabilidad de elevación de privilegios en Internet Explorer durante la validación de la instalación de archivos locales y durante la creación de claves de registro segura

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

5.3 – Gestión de vulnerabilidades
Common Vulnerability Score System (CVSS) - Ejemplo

- CVSS Severity (version 2.0):
  - CVSS v2 Base Score: 4.3 (MEDIUM)
  - Impact Subscore: 2.9
  - Exploitability Subscore: 8.6
- CVSS Version 2 Metrics:
  - Access Vector: Network exploitable; Victim must voluntarily interact with attack mechanism
  - Access Complexity: Medium
  - Authentication: Not required to exploit
  - Impact Type: Allows unauthorized modification

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
- 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
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
  - Implementación
  - Testing

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 comprueba 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 información diversa:
  - 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 hayas 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 informacion
  - Usuarios
  - Localización
  - IPs
  - Software
- Herramientas
  - stat
  - pdfinfo
  - file
  - https://metashieldclean-up.elevenpaths.com

5.4.2 – Pruebas de intrusión
Information gathering – Metadatos

- Fingerprinting Organization with Colleced Archives (FOCA)
- 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 sirve 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

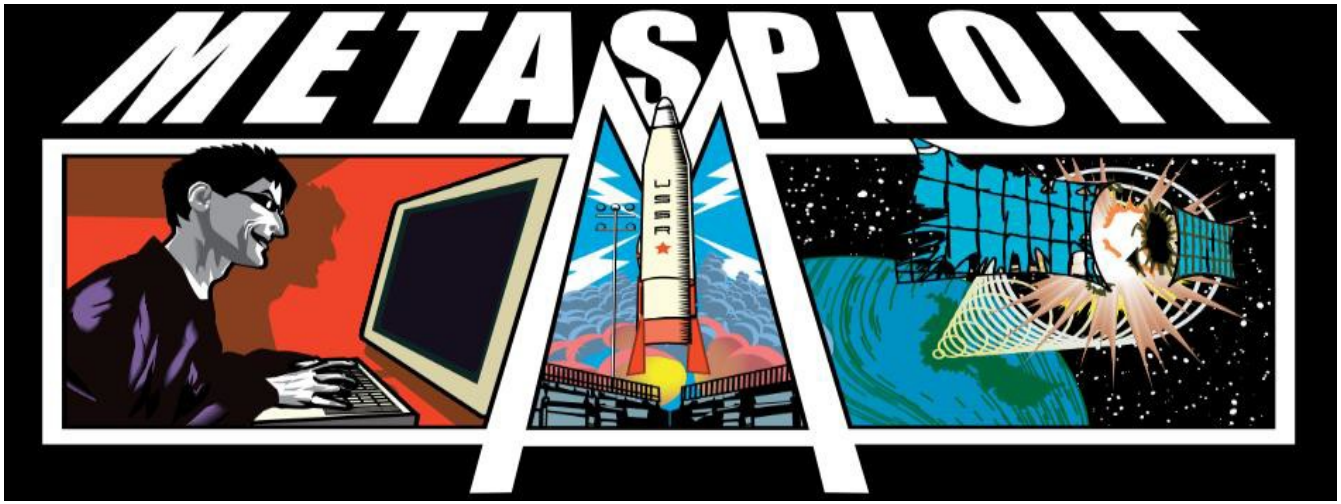
- Una vez detectadas las (posibles) vulnerabilidades se pueden explotar
- Hay que tener cuidado por varias razones
  - Hay cumplir el acuerdo con el responsable del sistema
  - Hay que evitar "tumbar" involuntariamente el sistema

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

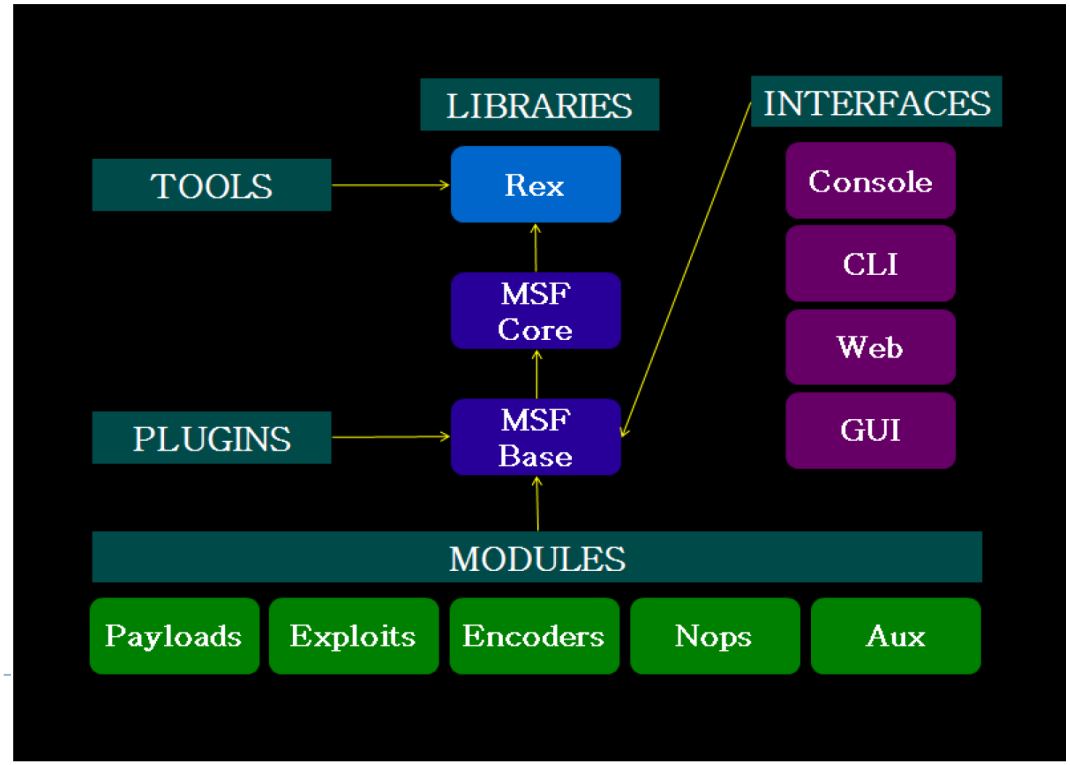
Explotación - Metasploit



- Una herramienta que permite
  - Desarrollar exploit
  - Ejecutar eploit 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