title: Clase 8 Proyecto 2016
Author: Einar Lanfranco, Claudia Banchoff, Matías Pagano, Diego Vilches
description: Asepctos básicos de seguridad web
keywords: Seguridad, owasp
css: proyecto.css

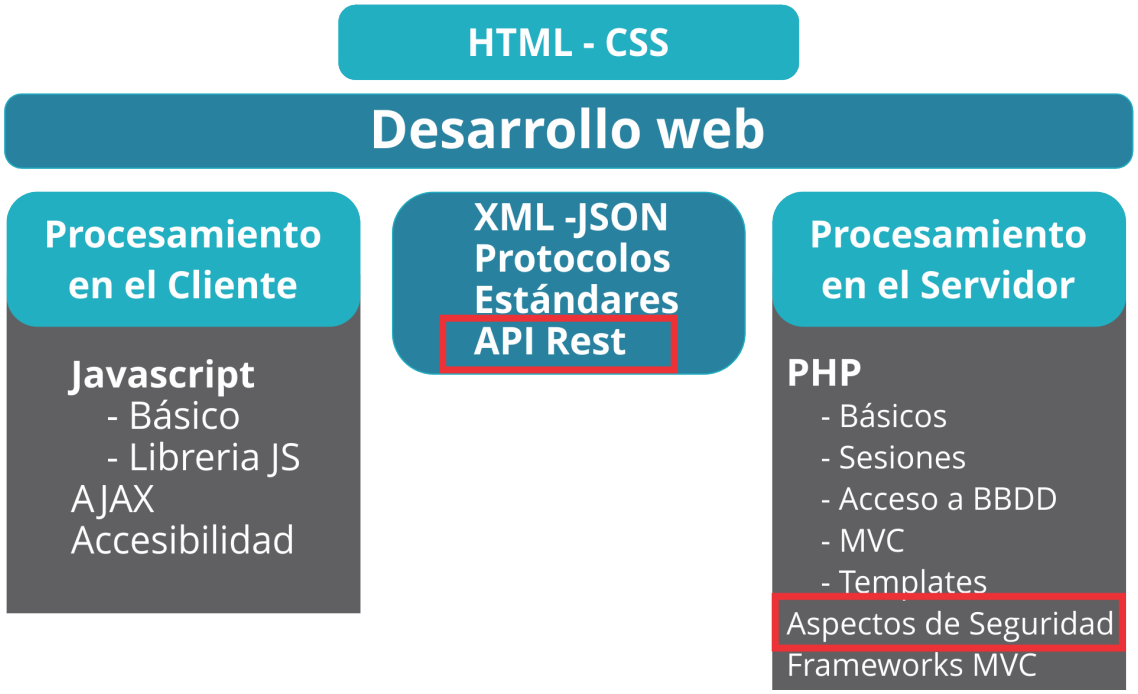
Proyecto de Software

Cursada 2016

---

data-rotate: 270

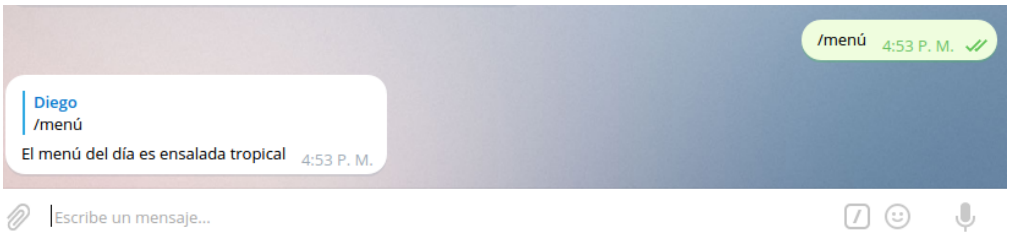
Hoy terminamos con ...



Temario

- **Repaso Clase Anterior**
    - Telegram Bot
    - API REST
- Aspectos básicos de seguridad web

Telegram Bot



REST

- Los sistemas que siguen los principios REST se los denomina también RESTful.
- Se basa en HTTP para intercambiar información.
- SIN estado.
- Se piensa en los recursos como una entidad que puede ser accedido públicamente.
- Ejemplo: misrecetas.com/api/v1/receta/chocotorta
- Cada objeto tiene su propia URL y puede ser fácilmente cacheado, copiado y guardado como marcador.

Recordemos que ....

Una petición HTTP consta de:

- Una URL y un método de acceso (GET, POST, PUT,...).
- Cabeceras. Meta-información de la petición.
- Cuerpo del mensaje (opcional).

Métodos HTTP

- GET: Usado para solicitar un recurso al servidor.
- PUT: Usado para modificar un recurso existente en el servidor.
- POST: Usado para crear un nuevo recurso en el servidor.
- DELETE: Usado para eliminar un recurso en el servidor

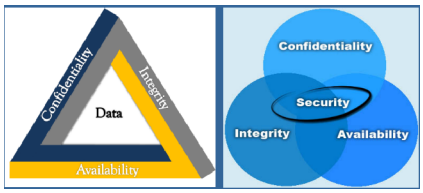
Seguridad en aplicaciones web

Algunos aspectos a considerar

Propiedades de la seguridad de la información

- **Confidencialidad:** Que sólo el que tenga acceso a la información lo pueda acceder.
- **Integridad:** Que la información no pueda ser modificada sin ser detectado.
- **Disponibilidad:** Que aquel que tiene acceso a la información siempre tenga la posibilidad de hacerlo.

Triángulo CID (CIA)



Contexto

- En sus inicios, la web permitía compartir información en forma libre.
- A medida que su uso se diversificó, agregando funcionalidad para acceder a información sensible, se hizo necesario contar con mecanismos que aseguren la privacidad y la integridad de la información.

---

class: destacado

Contexto

Una aplicación web insegura atenta no sólo contra la disponibilidad del servicio, la confidencialidad y la integridad de los datos, sino también contra la reputación de la organización.

En la vida real:

- El 22 julio de 2016 se conoce que Verizon acuerda pagar cerca de 5.000 millones de dólares por adquirir Yahoo.

En la vida real:

- Pero el 20 de septiembre de 2016 se conoce que hay 500 millones de cuentas comprometidas incluyendo: usernames, email, fecha de nacimiento, teléfonos, passwords, y preguntas secretas.

En la vida real:

- Y finalmente, el 2 de octubre de 2016 se conoce que Verizon, que había acordado comprar Yahoo por $4.8 Billion, ahora pide $1 Billion de descuento.

Aspectos de seguridad en PHP

- Algunas consideraciones exceden al desarrollador, ya que dependen de configuraciones de las redes o servidores.
- Otras, si dependen de una buena programación.
- **Nosotros ya vimos algunas ...**
    - ¿Cuáles?

El Proyecto OWASP

- OWASP: **Proyecto Abierto de Seguridad de Aplicaciones Web**
- Tiene por objeto ayudar a las organizaciones a desarrollar y mantener aplicaciones confiables.
- Brinda herramientas para aprender y realizar testeos, como Wescarb y Webgoat, foros, videos de ejemplos y mucha documentación, como la OWASP Guide, code Review.

OWASP Top Ten 2013:

- A1-- Inyección
- A2 - Pérdida de Autenticación y Gestión de Sesiones
- A3 - Secuencia de Comandos en Sitios Cruzados (XSS)
- A4 - Referencia Directa Insegura a Objetos
- A5 - Configuración de Seguridad Incorrecta
- A6 - Exposición de datos sensibles

- A7 - Ausencia de Control de Acceso a Funciones
- A8 -- Falsificación de Peticiones en Sitios Cruzados (CSRF)
- A9 - Utilización de componentes con vulnerabilidades conocidas
- A10 - Redirecciones y reenvios no validados

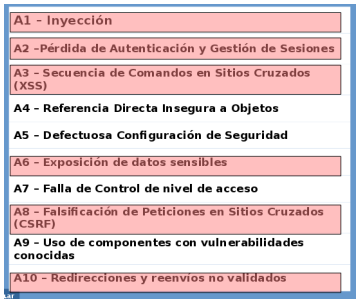
Note

- A1-- Inyección: Las fallas de inyección, tales como SQL, OS, y LDAP, ocurren cuando datos no confiables son enviados a un interprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al interprete en ejecutar comandos no intencionados o acceder datos no autorizados.
- A2 - Pérdida de Autenticación y Gestión de Sesiones: Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente, permitiendo a los atacantes comprometer contraseñas, claves, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios.
- A3 - Secuencia de Comandos en Sitios Cruzados (XSS): Las fallas XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar secuencia de comandos en el navegador de la víctima los cuales pueden secuestrar las sesiones de usuario, destruir sitios web, o dirigir al usuario hacia un sitio malicioso.
- A4 - Referencia Directa Insegura a Objetos: Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un fichero, directorio, o base de datos. Sin un chequeo de control de acceso u otra protección, los atacantes pueden manipular estas referencias para acceder datos no autorizados.
- A5 - Configuración de Seguridad Incorrecta: Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos, y plataforma. Todas estas configuraciones deben ser definidas, implementadas, y mantenidas ya que por lo general no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.
- A6 - Exposición de datos sensibles: Muchas aplicaciones web no protegen adecuadamente datos sensibles tales como números de tarjetas de crédito o credenciales de autenticación. Los atacantes pueden robar o modificar tales datos para llevar a cabo fraudes, robos de identidad u otros delitos. Los datos sensibles requieren de métodos de protección adicionales tales como el cifrado de datos, así como también de precauciones especiales en un intercambio
- A7 - Ausencia de Control de Acceso a Funciones: La mayoría de aplicaciones web verifican los derechos de acceso a nivel de función antes de hacer visible en la misma interfaz de usuario. A pesar de esto, las aplicaciones necesitan verificar el control de acceso en el servidor cuando se accede a cada función. Si las solicitudes de acceso no se verifican, los atacantes podrán realizar peticiones sin la autorización apropiada.
- A8 -- Falsificación de Peticiones en Sitios Cruzados (CSRF): Un ataque CSRF obliga al navegador de una victima autenticada a enviar una petición HTTP falsificado, incluyendo la sesión del usuario y cualquier otra información de autenticación incluida automáticamente, a una aplicación web vulnerable. Esto permite al atacante forzar al navegador de la victima para generar pedidos que la aplicación vulnerable piensa son peticiones legítimas provenientes de la victima.
- A9 - Utilización de componentes con vulnerabilidades conocidas: Algunos componentes tales como las librerías, los frameworks y otros módulos de sotware casi siempre funcionan con todos los privilegios. Si se ataca un componente vulnerable esto podría facilitar la intrusión en el servidor o una perdida seria de datos. Las aplicaciones que utilicen

componentes con vulnerabilidades conocidas debilitan las defensas de la aplicación y permiten ampliar el rango de posibles ataques e impactos.

- A10 - Redirecciones y reenvios no validados: Las aplicaciones web frecuentemente redirigen y reenvían a los usuarios hacia otras páginas o sitos web, y utilizan datos no confiables para determinar la página de destino. Sin una validación apropiada, los atacantes pueden redirigir a las víc=mas hacia si=os de phishing o malware, o u=lizar reenvíos para acceder páginas no autorizadas.

Riesgos Top 10



- Nosotros sólo vamos a ver algunas, las que están muy relacionadas con el desarrollo.
- Otras mencionadas acá están más relacionadas a la configuración de redes y aplicaciones.

---

class: destacado

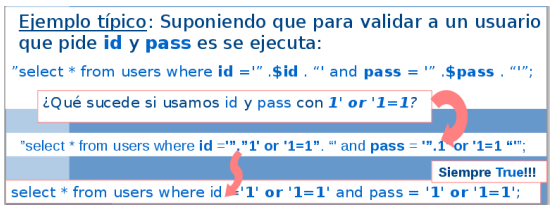
A1 - Inyección

Ocurre cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta.

- Este tipo de vulnerabilidad es **muy común**.
- El atacante envía simples cadenas de texto que explotan la sintaxis del intérprete atacado.
- Las fallas de inyección ocurren cuando una aplicación envía datos no confiables a un intérprete.
- Hay de distintos tipos, como SQL, LDAP, XPath, XSLT, entre otros.

Inyección SQL

- Una SQL Injection suele ocurrir cuando se arma en forma descuidada una consulta a la base de datos a partir de los datos ingresados por el usuario.
- Dentro de estos parámetros pueden venir el código malicioso.



A1- Inyección de comandos

- Por ejemplo, si tenemos una página que lista un directorio pasado por parámetro de la siguiente manera:
- **http://www.pepe.com/get.pl?usuario=pepe**
- En la app -> system("ls /home/$_GET['usuario']");
- Finalmente -> system("ls /home/pepe/");
- Incluyendo párametros maliciosos que permitan modificar lo que originalmente haría el comando, se podría listar otros directorios.

A1- Inyección de comandos

- Por ejemplo, incluyendo el **path ../** como parte del nombre del archivo que se pide, se puede realizar lo siguiente:
- **http://www.pepe.com/get.pl?usuario=pepe/../juan**
- En la app -> system("ls /home/$_GET['usuario']");
- Finalmente -> system("ls /home/pepe/../juan");
- Resuelto -> system("ls /home/juan");

A1- Inyección de comandos

- Si el intérprete es un shell, se podrían pasar otros comandos separados por **";"**
- **http://www.pepe.com/get.pl?usuario=pepe/../juan;rm -fr /tmp**
- En la app -> system("ls /home/$_GET['usuario']");
- Finalmente -> system("ls /home/pepe/../juan; rm -fr /tmp");
- Resuelto -> system("ls /home/juan; rm -fr /tmp");

Inyección ¿Cómo evitarlo?

- **Mantener los datos no confiables separados de comandos y consultas.**

    - APIs seguras que eviten el uso del intérprete completamente o provea una interfaz parametrizada.
    - Escapar los caracteres especiales utilizando una sintaxis de escape especial para dicho intérprete.
- Armar un esquema de permisos adecuado. Por ejemplo a nivel de base de datos.
- Realizar consultas concretas, evitar el "select * …"
- Utilizar Prepared Statement

Inyección ¿Cómo evitarlo?

- Filtrar los errores, por ejemplo a través del manejo de excepciones.
- Una vez puesto en producción, desactivar la variable error_reports de php.ini.
- Es importante buscar la forma específica para cada lenguaje, motor de base o mecanismo que se utilice para acceder a la base.
- Pero es DESEABLE que la aplicación se autoproteja → ¿Qué pasa si dependemos del php.ini cuando cambia de entorno nuestra aplicación?

DEMO

- **Ejemplo se SQLi:**
    - Sobre un versión de Joomla vulnerable
    - Utilizando una herramienta

---

class: destacado

A2 - Pérdida de autenticación

Si las funciones autenticación y gestión de sesiones son implementadas incorrectamente, es posible comprometer contraseñas, llaves, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios.

- El atacante utiliza fallas en las funciones de autenticación o gestión de las sesiones (por ejemplo cuentas expuestas, contraseñas, identificadores de sesión) para hacerse pasar por usuarios.
- Se aprovechan de vulnerabilidades en las secciones de cierre de sesión, gestión de contraseñas, tiempo de desconexión, función de recordar contraseña, pregunta secreta, actualización de cuenta, etc.

Pérdida de autenticación

Aspectos a tener en cuenta...

- Las credenciales deben estar protegidas.
- No debe ser posible sobreescribir o adivinar las credenciales: revisar funciones de cambio de contraseñas, recuperación de las mismas, etc.
- No mostrar id de sesión en la URL.
- Verificar siempre que se cierre la sesión o el tiempo en que caducan.
- No basarse sólo en que la sesión exista... poner controles adicionales como ser token de sesión, ip origen, revalidación ante funciones críticas, etc

---

Pérdida de autenticación

Dilema: Problema de usabilidad vs. seguridad

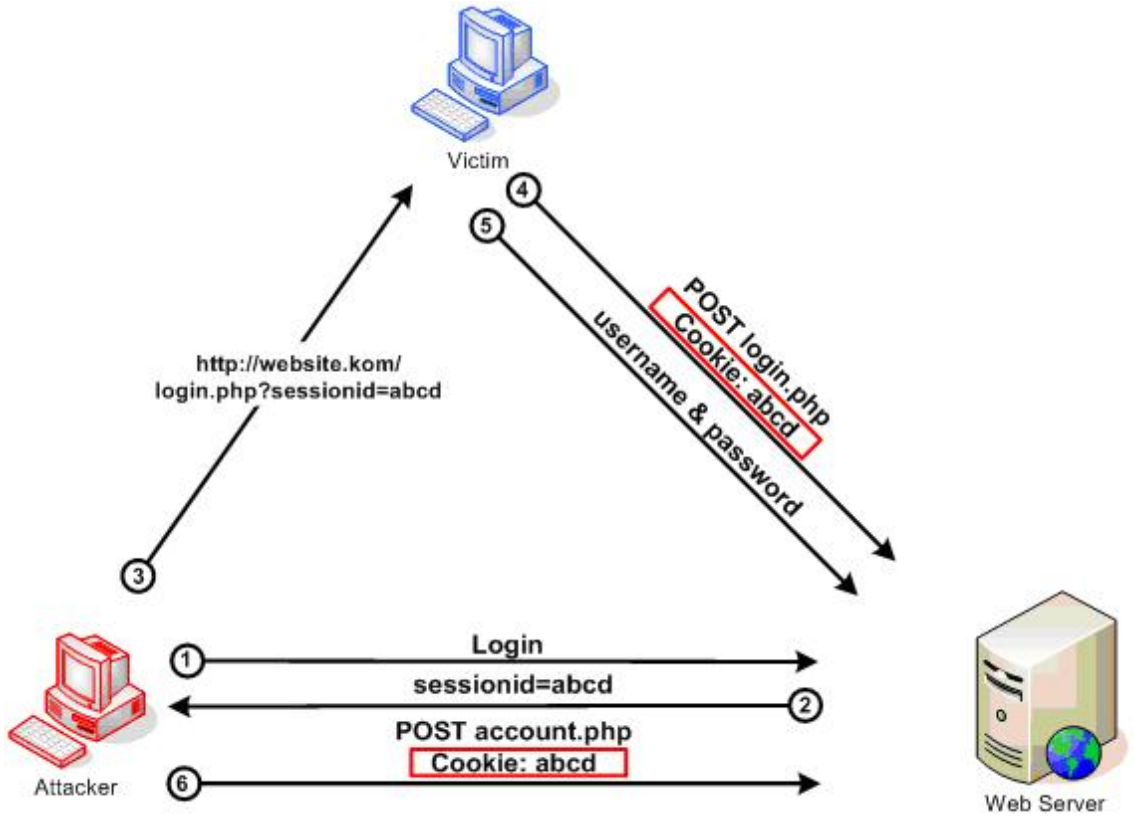
Problemas típicos Session

- **Ataques más comunes:**
    - Session Fixation
    - Session Hijacking
- Ambos ataques son similares, consisten en manipular datos y a través del id de sesión conseguir una sesión válida.
- **Sesión Fixation** es cuando el atacante puede fijar el id de sesión que usará el usuario legal
- **Sesión Hijacking** es robar el id de sesión, es cuando el atacante obtiene el id del usuario legal y la utiliza.

Session Fixation

- En este ataque lo que se hace es fijar la sesión, algo que no debería pasar ya que es el atacante quien dice a la aplicación que id de sesión utilizar y esta no se preocupa por verificarla.

Session Fixation



Demos

- **Veamos:**
    - Video HomeBanking
    - Demo Session Fixation

---

class: destacado

A3 - XSS - Cross Site Scripting

Ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada.

- Este tipo de vulnerabilidad también es **muy común**.
- Se lo conoce como XSS para que no sea confundido con CSS.
- En general ocurren cuando una aplicación toma datos de un usuario, no los filtra en forma adecuada y los retorna sin validarlos ni codificarlos.
- Puede insertarse HTML, Javascript, entre otros, a través de los formularios o la URL.

Cross site scripting

Tipos de XSS:

- **Existes 3 tipos de fallas de cross site scripting:**
    - Type-0, también conocido como XSS DOM based o XSS local.
    - Type-1, también conocido como XSS reflejado o no persistente.
    - Type-2, también conocido como XSS almacenado o persistente.

XSS - Cross Site Scripting

- Con esta vulnerabilidad es posible robar el acceso de los usuarios y violar la integridad y confiabilidad de sus datos.
- Por ejemplo robando nombres de usuarios, claves, cookies. También es posible ejecutar código en forma remota inyectando código a través de la URL.

XSS - Cross Site Scripting

- **Veamos dos ejemplos:**
    - **Del tipo almacenado: ver ejemplo del video y analizar**
        - ¿Qué pasa aquí?
        - ¿Cómo funcionaban las cookies y cual es su relación con las sesiones?
    - **Del tipo reflejado: ver ejemplo del documento de freebank**
        - ¿Por qué es más difícil de detectar?

XSS - ¿Cómo evitarlo?

- Validar la entrada: longitud, tipo, sintaxis, etc.
- Reemplazar las "", las palabras script, etc.
- Usar herramientas de detección de XSS en nuestra aplicación.
- Usar motores de templates como por ejemplo Twig.

---

class: destacado

A6 - Exposición de datos sensibles

El error más común en este área es simplemente no cifrar datos que deberían ser cifrados. Los atacantes normalmente no pueden romper el sistema criptográfico.

- Los datos sensibles deben tener protección extra como ser encriptación, tanto cuando están almacenados o en tránsito cuando se intercambia con el browser del cliente.
- Muchas aplicaciones no protegen correctamente los datos sensibles, como ser tarjetas de crédito, datos económicos y datos de autenticación.

Exposición de datos sensibles

- Un atacante no necesita atacar directamente, simplemente accede por ejemplo mediante un SQL injection y tiene a disposición todos los datos.
- Esta vulnerabilidad normalmente compromete todos los datos que deberían haber estado cifrados. Típicamente esta información incluye datos sensibles tales como datos médicos, cuentas de usuario, datos personales, tarjetas de crédito, etc.
- Si no hay cifrado en el transporte (típicamente https) cualquiera puede “escuchar” y “entender” la conversación entre cliente y servidor.

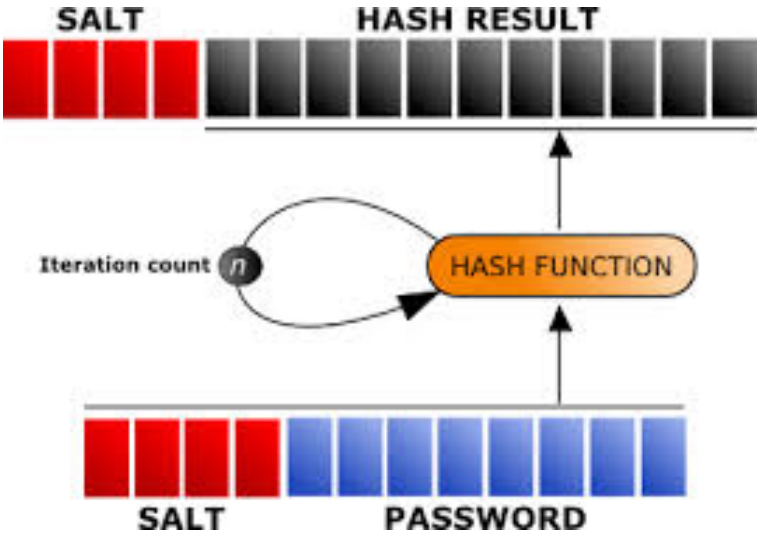
A6 - Exposición de datos sensibles

- **Posible escenario:**
    - Las passwords no usan salt -> las encriptaciones pueden ser precalculadas
    - Obteniéndose desde listas de password precalculadas o rainbow tables por comparación a partir del hash sin necesidad de calcularlas.

Ej: md5,sha1,etc → http://crackstation.net

A6 - Exposición de datos sensibles

¿SALT?



A6 - Exposición de datos sensibles

¿SALT?

SALT TYPE	SALTED PASSWORD	SHA256 HASH
unsalted	passwd!	12f225551a043b6e136b2cf03546b06efb289d29ab42cebfd78ee101d8555304
Prefix	12345passwd!	b38c49760d484119f227fab640cb1415d58f765c0727dc9ad7e0a5a66003d041
Infix	pass12345wd!	a9fa7c693b691c8ceded848877afdb1259f28f194863ebb33c07c4bbfa1ff04c
Postfix	passwd!12345	1315abea2576c3b6270712df587359d614ae1a1698d69dd2175459e411f678f5

A6 - Ejemplos reales:

- 4/4/16 -> 49,611,709 records de datos personales de ciudadanos turcos expuestos
- X/X/15 -> 21.5 + 4.2 millones de datos oficiales de USA robados
- 1/7/15 -> 36 millones de cuentas de Ashley Madison expuestas
- 10/10/14 -> 5 millones de cuentas de google expuestas
- 04/11/14-> iTunes accounts 3,449,726
- 01/01/14-> Roban datos de 40 millones de tarjetas de crédito a TARGET
- 12/7/12 → 450000 cuentas de yahoo publicadas
- 9/5/12 → 55000 cuentas de Twitter en Pastebin

- 6/6/12 → 6.5 millones de cuentas de Linkedin expuestas → pero SHA-1

A6 - Exposición de datos sensibles

- **¿Estos casos les parecen raros?**
    - Ver https://haveibeenpwned.com/
    - Ver https://breachalarm.com/
    - Ver https://www.trustify.info/check

Exposición de datos sensibles

¿Cómo evitarlo?

- Siempre cifrar datos críticos → nadie sin los permisos adecuados debería leerlos.
- Estos cuidados deben aplicarse tanto cuando el dato esta en el cliente, como cuando esta en tránsito o almacenado en el servidor.
- En el caso de contraseñas NADIE debería leerlos → tienen que almacenarse como hash y en el momento de la comprobación comparar hash.
- Utilizar funciones estándar y de ser posible es deseable usar SALT.
- Bcrypt es el metodo por defecto de la función password-hash de php desde 5.5 . http://php.net/manual/es/function.password-hash.php

---

class: destacado

DEMO

Veamos como funciona un Sniffer en un sitio con https y otro con http

---

class: destacado

A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)

Ocurre cuando un atacante pide a una aplicación datos a nombre de otro y ésta toma ese pedido como una petición válida.

A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)

- También conocido como XSRF, CSRF, y Cross Site Reference Forgery, consiste en explotar la confianza que el sitio tiene en el usuario.
- Los atacantes crean peticiones HTTP falsas.
- Engañan a la víctima al enviarlas a través de etiquetas de imágenes, XSS, o muchas otras técnicas. Si el usuario está autenticado entonces el ataque será exitoso.
- Los atacantes pueden cambiar cualquier dato que la víctima esté autorizado a cambiar, o acceder a cualquier funcionalidad que la víctima esta autorizado a realizar.

Falsificación de Peticiones en Sitios Cruzados (CSRF)

- Sí las acciones de un sitio son urls fijas, como por ejemplo:

http://my_banco/transferir?cantidad=1000&cuentadestino=123323

- Es posible que un tercero ejecute la misma a nombre de la víctima.
- Típicamente un atacante puede embeber código malicioso HTML o Javascript en un mail o un sitio web para requerir una tarea específica a través de una url y se ejecutaría sin el conocimiento del usuario, directamente u utilizando una falla de Cross-site Scripting.

¿CSRF = XSS?

- **Funcionan exactamente al revés:**
    - Los usuarios generalmente confían que el contenido mostrado en sus navegadores es lo que el sitio web visitado quiere realmente presentar al usuario. El sitio web asume que si una acción requerida fue ejecutada, es lo que el usuario quizo ejecutar intencionalmente.
- Cross-Site Scripting explota la confianza que el cliente tiene en el sitio web o la aplicación.
- CSRF explota la confianza que el sitio tiene en el usuario.

---

class: destacado

DEMO

Veamos un ejemplito para enviarle a sus amigos!

Mejores Patrones

Token: Usualmente se maneja en un campo oculto en la página y es generado dinámicamenteen en forma random para cada link que aparezca en la página html. Pueden ser uno o varios. Pasos:

1. Se genera una lista de IDs únicos antes de enviar la página al usuario.

2. Cada ID es agregado a cada link o form antes de enviarse al usuario.
3. Manetener la lista de IDs en la sesión del usuario para verificar los HTTP request que se reciben. Si el ID es válido entonces es un requerimiento válido
4. Si el ID no esta presente, terminar la sesión del usuario y mostrar el error al usuario.

Mejores Patrones

```
<form action="/transfer.do" method="post">
<input type="hidden" name="CSRFToken"
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEwYzU1YWQwMTVhM2J
mNGYxYjJiMGI4MjJjZDE1ZDZjMTViMGYwMGEwOA==">
  …
  </form>
```

Mejores Patrones - Interacción del usuario:

- Generar verificaciones ante algunas actividades, como ser transferencia de fondos, esto puede ser alguna pregunta o reautenticacion del usuario.
- Con esto evitamos cualquier ataque CSRF, ya que el atacante no conoce las credenciales y si lo hace, no es necesario el CSRF.
- Hay que tener cuidado como se usa porque afectamos la funcionalidad

---

class: destacado

10 - Redirecciones y reenvíos no validados

Ocurre cuando una aplicación redirige o reenvía a los usuarios hacia una página o sitio web sin validación adecuada. En este caso, los atacantes pueden redirigir a las víctimas hacia otros sitios sin que el usuario se percate de la situación.

- Un atacante crea enlaces a redirecciones no validadas y engaña a las víctimas para que hagan clic en dichos enlaces. Las víctimas son más propensas a hacer clic sobre ellos ya que el enlace lleva a una aplicación en la que se confía.
- Si la página de destino se especifica en un parámetro no validado, se permite a los atacantes elegir dicha página.

Redirecciones y reenvíos no validados

- Ejemplo típico:

http://www.example.com/redirect.jsp?url=evil.com

- Si el atacante, manipula el parámetro url, puede redirigir hacia otro sitio.
- **¿Como puedo evitar esto?**
    - Evitando el uso de redirecciones y reenvíos.

- Validar entradas y parámetros.

En resumen

- VALIDAR SIEMPRE entradas de datos.
- Usar APIs conocidas y probadas.
- Estar atentos a situaciones que involucren uso de eval(), include(), requiere(), fopen(), system(), etc...
- Revisar aspectos de configuración básicos tanto en el intérprete, framework, web server o base de datos.
- Usar htmlentities(), urlencode(), etc.
- Usar $_GET, $_POST, etc. según corresponda.