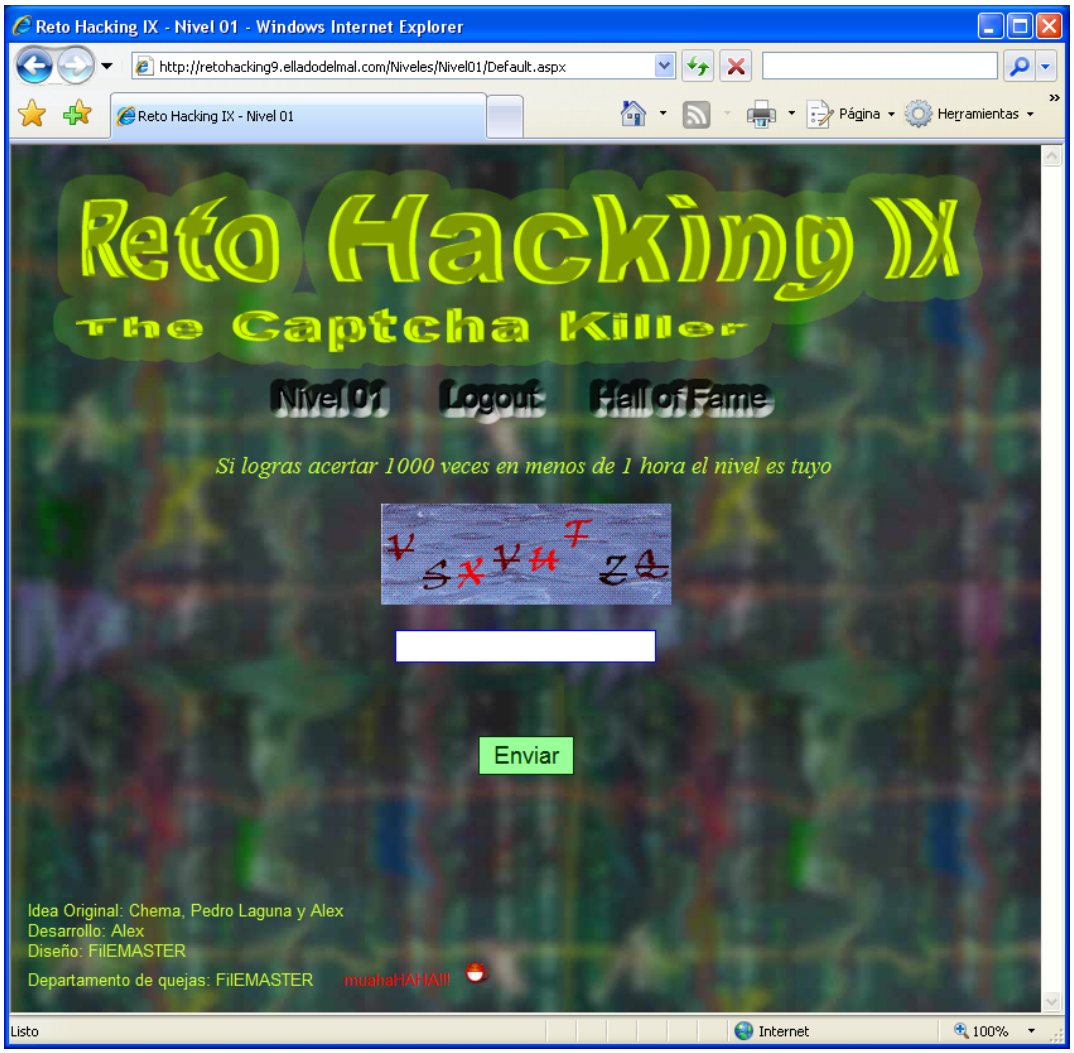
Solución al Reto Hacking IX de Informática 64

Septiembre 2008 © Daniel Kachakil

Introducción

Este documento describe una solución al Reto Hacking IX de Informática 64, el cuarto de la segunda temporada, que se publicó el 12 de septiembre de 2008 en la siguiente dirección web:

`http://retohacking9.elladodelmal.com`



Pistas

De nuevo nos encontramos ante un reto en el que no hacen falta pistas, puesto que el objetivo estaba claro desde el principio. En todo caso, el título del post que anunciaba el reto (*Hay que ser muy hombre o muy mujer*) tenía mucha relación con el tema, en el sentido de que necesitaremos habilidades humanas para superar el reto, pero eso se ve a posteriori y tampoco sirve de ayuda.

Visión general del reto

Tras darnos de alta y acceder con nuestro usuario registrado, nos encontramos ante el primer nivel de los 10 que forman el reto. Por el texto que aparece en cada nivel y por el propio título del reto podemos intuir que la cosa va de acertar captchas, nada menos que 1000 por cada nivel y en un plazo máximo de una hora que se empezaba a contar desde el primer acierto.

Con la habilidad suficiente tal vez no fuera imposible lograrlo a mano (con un promedio de 3,6 segundos por acierto), pero evidentemente se trataba de automatizar las peticiones, ya que hacerlo a mano resultaría un tanto aburrido y lento.

De todas formas, creo que en el planteamiento del reto no se tuvo en cuenta que prácticamente todos los niveles tenían una solución alternativa bastante más sencilla (o a lo mejor me equivoco), haciendo que el nivel 1 tuviera una solución idéntica a la del nivel 8. Por ello, explicaré la solución "oficial" por un lado y el atajo por otro (me imagino que éste último es el que utilizamos la mayoría de los participantes).

Nivel 1

Si le echamos un vistazo al código fuente de la página, veremos que el botón *Enviar* tiene asociada una pequeña función JavaScript llamada *ValidateCaptcha*, cuyo código aparece un poco más arriba. Simplificándola un poco, era similar a esta:

```
function ValidateCaptcha() {
    if (document.getElementById('txtCaptcha').value == 'VSXVUTZQ') {
       return true;
    } else {
       return false;
    }
}
```

Bastaba con deshabilitar los scripts para que cada pulsación del botón *Enviar* contara como un acierto más. Además, la solución del captcha aparece en el propio código de la página como texto en claro, por lo que podríamos automatizar su lectura y su posterior envío a través del formulario.

Supongo que en este caso la solución "oficial" coincide con el atajo, es decir, capturar y reenviar 1000 veces la misma petición, manteniendo los valores asociados a los parámetros relevantes que se enviaban por post y la cookie de autenticación:

```
__VIEWSTATE = /wEPDwUKMTEwNjI0MjY0MQ9kFgJmD2QWAgIDD2QWAgIB ...
__EVENTVALIDATION = /wEWBAKt2PnGBQKsoYL1DwK15qaS ...
ctl00$ContentPlaceHolder1$btEnviar = Enviar

.ASPXAUTH = 4A4AD74485F839D53AC26F746FCBA03A14F63307CC4971 ...
```

Evidentemente, cada cual opta por las herramientas con las que se siente más cómodo, así que hay cientos de formas de automatizar esta tarea. En mi caso adapté parte del código que ya tenía hecho hace mucho tiempo y que incorporé en Marathon Tool (principalmente la función *Download* de BaseEngine.vb). Lo importante es que, transcurridos unos minutos dedicados a enviar las 1000 peticiones necesarias, pasamos automáticamente al siguiente nivel.

Nivel 2

Muy similar al nivel anterior, en esta ocasión nos encontramos con un campo oculto con la solución al captcha:

```
<input name="ctl00$ContentPlaceHolder1$_valor" type="hidden"
   id="ctl00_ContentPlaceHolder1__valor" value="VSXVUTZQ" />
```

De nuevo, podíamos haber leído y reenviado el valor que se almacenaba en este campo, pero bastaba con repetir exactamente el mismo procedimiento del nivel 1, evitando el envío del parámetro *ctl00$ContentPlaceHolder1$_valor* (o en general, haciendo que coincida con el valor de *ctl00$ContentPlaceHolder1$txtCaptcha*)

Nivel 3

En esta ocasión, por ejemplo para el captcha "UWPCTNUU", nos encontramos con una cookie como la siguiente:

```
valor = 55575043544e5555
```

Es evidente que se trata de los valores ASCII de cada carácter del captcha, puestos en valor en hexadecimal y concatenados. Por ejemplo, en la secuencia anterior tenemos que 0x55 = "U", 0x57 = "W", 0x50 = "P", etc.

Podíamos optar por leer y convertir estos valores en sus correspondientes caracteres para decodificar el captcha de forma automatizada, pero de nuevo bastaba con repetir exactamente el mismo procedimiento del nivel 1, enviando además una cookie llamada *valor* con su contenido en blanco.

Nivel 4

Este nivel puede parecer incluso más fácil que el anterior, puesto que el nombre de la imagen corresponde con el valor del captcha mostrado:

```
<img id="ctl00_ContentPlaceHolder1_ImgCapcha" src="imagenes/DVNZVSHC.gif" />
```

Sin embargo, se trata del primer nivel que realmente obliga a la lectura de este valor y a su posterior reenvío, por lo que tras realizar una petición inicial, procesaremos el contenido de la página para extraer el captcha (por ejemplo, buscando la posición de la primera aparición de la cadena ".gif" y recuperando los 8 caracteres anteriores). En la petición siguiente lo enviamos por POST en *ctl00$ContentPlaceHolder1$txtCaptcha*, leyendo de paso el nuevo captcha generado y así sucesivamente hasta llegar a completar las 1000 peticiones (sin olvidarnos de la cookie *ASP.NET_SessionId*).

Nivel 5

En este nivel observamos que hay un parámetro en la URL llamado *key*. Su longitud coincide con la de los captchas que han ido apareciendo hasta el momento:

```
http://retohacking9.elladodelmal.com/Niveles/Nivel05/Default.aspx?key=ONOATHZK
```

Podemos molestarnos en intentar averiguar la relación entre esa clave y el correspondiente captcha generado, pero es totalmente innecesario, ya que de nuevo podemos reenviar 1000 veces cualquier petición válida y superaremos el nivel…

De todas formas, supongo que la solución oficial era la primera, así que la expongo aquí porque la relación es muy fácil de obtener. Por ejemplo:

```
Key = AAAAAAAA -> Captcha = OAPSUNMO
Key = BBBBBBBB -> Captcha = PBQTVONP
Key = CCCCCCCC -> Captcha = QCRUWPOQ
```

Es evidente que existe una relación lineal entre cada carácter, por lo que basta con desplazar el alfabeto las siguientes posiciones respectivamente (es decir, se trata de un clásico y simple cifrado de Vigenère):

```
14, 0, 15, 18, 20, 13, 12, 14
```

Nivel 6

En este nivel nos encontramos con un conjunto de captchas con nombres de algunos dioses de la mitología griega. Bastaba con actualizar o refrescar la página una y otra vez para darnos cuenta de que el conjunto de nombres siempre aparecía de forma secuencial. En caso de intentar acertarlos, de vez en cuando se rompía la secuencia ante algún fallo, pero en general se respetaba el orden siempre que se acertaba el captcha.

Por tanto, si tomamos nota de un trozo de la secuencia lo suficientemente largo como para obtener una tasa de aciertos razonable, podremos superar el nivel sin mayor dificultad, a pesar de requerir una cantidad de tiempo y peticiones considerablemente mayor al estrictamente necesario de haber tomado nota de la secuencia completa. Por ejemplo, con la siguiente secuencia sería más que suficiente:

```
Himeneo, Cibeles, Evadne, Cerbero, Hipotoo, Hebe, Irene, Hipolita, Argos,
Alcmena, Hermes, Adonis, Ganimedes, Furias, Calisto, Afrodita, Faetonte, Gigantes
```

La clave de este método consiste en detectar el primer acierto (por ejemplo, buscando la cadena "aciertos" en la respuesta) y sincronizar la secuencia a partir de ahí. Iremos realizando peticiones con el primer valor de la secuencia hasta que se detecte un acierto. Entonces incrementamos la posición actual de la secuencia hasta encontrar un fallo, momento en el que volveremos a empezar por el principio y así sucesivamente hasta completar los 1000 necesarios. Como curiosidad, tras el último elemento de la lista el servidor devolvía un error 500 (posiblemente por algún índice fuera de rango).

Nivel 7

Este nivel parece idéntico al tercero (el texto del captcha se encuentra codificado en la cookie "valor"), pero se diferencia por un pequeño detalle que nos va a complicar un poco la solución. Después de acumular 6 aciertos (ya fueran consecutivos o no), nos encontraremos con la siguiente frase que nos impedirá avanzar:

Ya llevas demasiados aciertos seguidos, ¿no te parece?

De alguna forma, la aplicación era capaz de detectar si se estaban recibiendo demasiadas peticiones válidas desde una misma ubicación. Hay muchos factores que se podrían haber tomado como referencia para identificar dicha repetición, así que mis primeros intentos se centraron en enviar peticiones incorrectas, en cerrar y abrir de nuevo la sesión del usuario, en dejar pasar un buen rato entre aciertos, en modificar parámetros de la petición que enviaría el propio navegador (como *User-Agent*, *Referer*, *X-Forwarding-For*, etc.), pero todas esas pruebas no tuvieron éxito. También hice algunas pruebas desde otra IP, pero tampoco hubo manera de burlar el mecanismo.

Resulta que desde la URL original del reto este bloqueo es permanente y no hay ninguna forma de evitarlo, de ahí que nos enganchábamos durante tanto tiempo. Sin embargo, tras sugerir una revisión del nivel por mi parte, se detectó el fallo al día siguiente y se corrigió varias horas más tarde, habilitando una URL alternativa cuya única misión es la de permitir la superación de este nivel:

```
http://retohackingIX.elladodelmal.com:81
```

La clave estaba en realizar una petición desde otra IP después de cada 6 peticiones correctas. Para ello podemos buscar y usar cualquier proxy anónimo que funcione y configurarlo para ser usado en nuestra siguiente petición si la respuesta contiene la cadena "demasiados aciertos", por ejemplo. Por lo demás, como el nivel 6.

Nivel 8

Otro nivel curioso, puesto que nos devuelve en un altísimo porcentaje de peticiones (por no decir en todas ellas) una página de error con el texto:

Bad Request (Invalid URL)

Por lo demás, parece que tiene alguna similitud con el nivel 5, puesto que también existe un parámetro en la URL llamado *key*, aunque en esta ocasión vemos caracteres extraños (la gran mayoría no son alfanuméricos). Si eliminamos este parámetro de la petición, se genera uno nuevo de forma aleatoria.

El caso es que si modificamos este parámetro a nuestro antojo (usando caracteres validos), podremos llegar a visualizar un captcha. El problema es que nos encontraremos con caracteres raros (o no imprimibles) en el propio captcha y no podremos acertarlo, así que tendremos que escoger una clave adecuada (básicamente al azar) que nos muestre un captcha que podamos acertar. Por ejemplo:

```
Key = ssssssss -> Captcha = JJ@0175@
```

Una vez localizada una petición válida, únicamente tendremos que repetirla 1000 veces para superar el nivel. Desconozco cual era la solución oficial, pero por más que averigüemos la relación existente entre la clave y el captcha, no tendría sentido intentar aplicar la solución porque ya se nos están rechazando todas las peticiones aleatorias que genera la propia aplicación del reto, así que la respuesta tampoco pasaría el filtro del "Bad request" por muy válida que fuera.

Nivel 9

En este nivel tampoco se observa nada raro a primera vista. El nombre del fichero correspondiente a la imagen de cada captcha tiene una longitud de 32 caracteres, aparentemente en formato hexadecimal. Sin embargo, dicho nombre no parece tener ninguna relación con el contenido del captcha. Durante la recogida y el análisis de los datos, es fácil comprobar que de vez en cuando se repiten algunos captchas, pero no su secuencia de aparición (como ocurría en el nivel 6).

Viendo que no había por dónde cogerlo, solo se me ocurrió lanzar un ataque de diccionario al que iba añadiendo nuevas entradas de forma manual mientras se iban realizando peticiones buscando algún valor conocido y lanzando su respuesta. A medida que iba pasando el tiempo, el diccionario tendría más entradas conocidas y, por tanto, la efectividad del método iría en aumento y podría resolverse en menos de una hora.

Al final no llegué a introducir más de 40 entradas en el diccionario, porque la probabilidad de acertar ya era más que suficiente como para superarlo, así que esto no es tan costoso como puede aparentar. El diccionario podría tener este aspecto:

```
014d88d2d892431bb5855ede7cb87da3 -> NPPCSJYG
04bad533ac664836b01512e06cfb78d6 -> NBEGLXZO
0754db6e0a3c40c78e457026f0e9d933 -> IEPUMSMY
053a30e818824e1e9cfc5b0eaf1a1641 -> DTSSBWHQ
035b1ecb11fc4849ac916e60c70bad2b -> SQIGGOUZ
053721ce60114feba691036713355399 -> DCIJMTCT
```

De todas formas, la idea de la solución oficial era un tanto más complicada, puesto que en el servidor existía un fichero XML con la relación de cada imagen con su correspondiente captcha y el parámetro en el que introducíamos la respuesta era vulnerable a XPath injection. A pesar de contar con esta pista (que me facilitó Chema tras finalizar el reto, para ver si era capaz de sacarlo así), la solución del nivel tampoco me resultó tan sencilla, tal y como explicaré a continuación.

En primer lugar, intentaríamos las típicas inyecciones que devuelven siempre un valor verdadero, pero esto no sirve de nada, puesto que devuelve el primer nodo:

```
' or 1=1 or ''='
```

Sin embargo, si acertamos el captcha en la consulta, sí que conseguimos diferenciar el caso verdadero del falso. Por ejemplo:

```
SZMBGGNE' and 1=1 or 'a'='
SZMBGGNE' and 0=1 or 'a'='
```

Esto plantea un pequeño problema a la hora de automatizar cualquier método y es que para que nos dé por válido un captcha, tendremos que acertarlo a mano previamente. Y para eso mejor no inyectamos nada y acabamos antes, ¿verdad? No obstante, ahora tenemos un mecanismo que nos permite preguntar acerca de la estructura interna del XML, aunque tenga que ser a mano. Por ejemplo, podemos probar con las siguientes inyecciones (por cierto, todas estas dan un resultado positivo):

```
HGZDGXFF' and count(//child::*)>10 or 'a'='
GVMOMWYD' and count(//child::*[position()=1])>0 or 'a'='
JYQLCIJO' and count(//child::*[position()=1]/node()[position()=1])=2 or 'a'='
MFIYFUCZ' and count(//child::*[1]/node()[1]/node()[1])=1 or 'a'='
GGULDIES' and string-length(//child::*[1]/node()[1]/node()[1])=36 or 'a'='
```

Tal vez la inyección más relevante sea la última, puesto que nos está indicando que el texto del nodo tiene una longitud de 36 caracteres (es decir, incluye la extensión del fichero). A partir de inyecciones como estas, podemos intuir que la estructura del XML subyacente debería ser similar a esta (sin conocer los nombres de las etiquetas):

```
<captchas>
    <captcha>
        <imagen>014d88d2d892431bb5855ede7cb87da3.gif</imagen>
        <valor>NPPCSJYG</valor>
    </captcha>
    <captcha>
        <imagen>04bad533ac664836b01512e06cfb78d6.gif</imagen>
        <valor>NBEGLXZO</valor>
    </captcha>
    <!-- ... -->
</captchas>
```

Seguramente, el algoritmo implementado se basaba en que el servidor elegía un fichero al azar (preexistente) dentro de la carpeta de imágenes y comprobaba si dicho nombre coincidía con el resultado de una consulta XPath similar a esta:

```
"/captchas/captcha[@imagen='" + txtCaptcha.Text + "']/text()"
```

Así que, tras realizar varias pruebas sin éxito, finalmente conseguí una inyección bastante simple que dependía únicamente del nombre del fichero del captcha, por lo que ya se puede resolver de forma automatizada (casi idéntica al nivel 4):

```
' or text()='037a021d3e31419ca9e151508375456a.gif' or 'a'='
```

Nivel 10

El último nivel del reto no tenía nada que ver con los anteriores, porque nuestro objetivo era completamente diferente. No se trataba de acertar captchas, sino que nos mostraba una secuencia de 3 parejas de manos, cada una indicando un número del 1 al 5 con los dedos. Teníamos que acertar la contraseña 5 veces (no necesariamente consecutivas), sin importar los fallos que cometiéramos, por lo que más de uno superó este nivel sin saber realmente cómo lo hizo (y yo me incluyo, jeje) ;-)

Sin pistas podíamos hacer cientos de pruebas diferentes, pero al final solo teníamos que introducir el valor numérico de la mano que aparecía en primer lugar (la mano izquierda de la primera tanda) y nos daba la respuesta por válida. Al parecer, esto estaba basado en la escena del guardia de la parte 2 del juego Monkey Island 2.

Agradecimientos y comentarios

No quiero terminar este documento sin poner de manifiesto algunos aspectos que, en mi opinión, deberían mejorarse en próximos retos. Todo esto dejando de lado el fallo del nivel 7 (a pesar de que me quitó toda la ventaja que tenía por haber llegado el primero), el del nivel 8 (¿estaba previsto el *Bad request*?), el del nivel 9 (cuando me topé con él no cuadraban las imágenes de los captchas con el texto esperado) e incluso de los niveles cuya resolución era idéntica y hacía que el reto fuera algo monótono (aunque me pueda equivocar, para mí que todo eso no estaba previsto).

Me refiero sobre todo a los errores que se cometieron en la gestión de cuentas de usuario, puesto que las cuentas se bloqueaban con tan solo 5 intentos fallidos y el recordatorio de contraseña solo pedía el nombre de usuario (que además era público y conocido, puesto que aparecía en el *Hall of fame*). Además de recibir varios correos electrónicos con recordatorios de contraseñas que nunca solicité, también me encontré con que mi cuenta de usuario estaba bloqueada justo cuando se corrigió el nivel 7 (y me consta que no fui el único afectado). Está claro que no todo el mundo jugaba limpiamente y esto se tenía que haber previsto, porque eso es un ataque de denegación de servicio en toda regla y si un juego deja de ser jugable mosquea un poco…

Dada la complejidad intrínseca de la contraseña autogenerada, no hacía falta bloquear la cuenta en ningún caso y tampoco fue buena idea poner un recordatorio sin utilizar mecanismos de protección como el de la pregunta secreta. Y no será porque cueste programar estos detalles, porque usando el *membership* de ASP.NET 2.0 esto se puede lograr declarativamente, sin ninguna línea de código.

Por otro lado, tampoco entiendo cómo es posible que el *Hall of fame* haya empeorado con respecto al de retos anteriores. ¿Dónde está la fecha y hora de finalización de los diferentes niveles (¡por lo menos la del último!)? ¿Dónde está el medio punto que obtendría el primero en superar cada uno de los niveles? Incluso el orden en el que aparecían los ganadores era incorrecto hasta el 4º día (y lo sigue siendo, porque tampoco respeta estrictamente el orden de finalización)

En fin, tampoco quiero que parezca que estoy tirando por tierra el trabajo de nadie, ni mucho menos, porque me consta que estas cosas cuestan más de lo que parece, pero es una auténtica lástima que parte del reto se eche a perder porque se descuidan detalles importantes o porque se pasan por alto algunos fallos que se podían haber evitado invirtiendo algo de tiempo en probar el reto de principio a fin.

De todas formas, como una cosa no quita la otra, quiero agradecer el trabajo y el esfuerzo de todo el equipo de Informática 64 que ha colaborado en este reto (y espero que ninguno se mosquee por mis comentarios, que son críticas constructivas, ¿ok?).

Por cierto, la solución del reto también era muy factible usando técnicas de reconocimiento de caracteres. Aprovechando el parón del nivel 7, más de uno nos curramos un algoritmo de OCR, pero no me quiero enrollar más… ¿Os animáis a intentarlo de esa manera? ¿publicará su solución nuestro amigo "bambú"? ;-)

Saludos,

Daniel Kachakil
dani@kachakil.com