Solución al Reto Hacking X de Informática 64

Septiembre 2009 © Daniel Kachakil y Román Medina-Heigl Hernández

Introducción

Este documento describe una solución al Reto Hacking X de Informática 64, que se publicó el 24 de julio de 2009 en la siguiente dirección web:

`http://retohackingx.elladodelmal.com`

Pistas

En esta ocasión únicamente contábamos con la descripción general del reto, publicada un día antes de su comienzo, en el que aparecen algunos párrafos indicando que el reto trata de lo que se ha hablado en el último año en el blog, que hay que seguir un camino claro siendo algo imaginativo y poco más.

En nuestra opinión, ninguna pista nos orientaba claramente en ninguna de las fases, así que no nos detendremos más con las pistas y vamos directamente a su resolución.

Nivel 1

Tras darnos de alta y acceder con nuestro usuario registrado, nos encontramos ante el primer nivel, en el que vemos a Homer Simpson en la puerta del *Leftorium*, la tienda de Flanders. Si pulsamos con el ratón sobre el teclado rojo de la parte izquierda, nos aparecerá un cuadro de autenticación típico (usuario y contraseña).

Inspeccionando el código fuente de la página no encontramos nada raro, así que todo parece indicar que se trata de acertar el usuario y la contraseña. Tras pasar un buen rato intentando todo tipo de inyecciones, combinaciones de usuarios y contraseñas más o menos absurdas (nombres y palabras relacionadas con los Simpson, contraseñas típicas, etc) o buscando esteganografía en las imágenes, no conseguimos obtener ningún indicio, ningún mensaje ni respuesta diferente de "Acceso denegado".

Al final, la solución resultó ser una sencillísima inyección SQL (bastante peculiar, eso sí) en el campo de la contraseña. Resulta que los textos que aparecían en la imagen eran pistas que teóricamente nos debían hacer pensar en que la página estaba diseñada por un zurdo (Flanders) y que los zurdos hacen las cosas "al revés". Por tanto, en vez de programar una consulta SQL típica, un zurdo habría hecho algo así:

```
"SELECT * FROM Usuarios WHERE '" + txtContraseña + "' = Contraseña"
```

Llegados a este punto de extrema imaginación, simplemente tenemos que inyectar una cadena de texto que nos permita saltarnos la autenticación. Para ello no hay más que invertir la clásica inyección de bypass:

```
' or '1'='1  →  1'='1' or '   ( o equivalentemente:  '='' or ' )
```

Con esta inyección, conseguiremos una consulta bien formada cuya primera condición es siempre verdadera y que al ejecutarse sobre el motor de bases de datos nos devolverá todos los registros de la tabla en cuestión:

```
"SELECT * FROM Usuarios WHERE '1'='1' or '' = Contraseña"
```

Con la solución delante de nuestras narices la cosa parece muy fácil (y en el fondo lo es), pero en realidad podría haber sido una fase de tipo acertijo o de cualquier otra tecnología. Nada nos hacía pensar en que la autenticación estaba basada en una consulta SQL (podría haber sido también XPath, LDAP, ... o incluso "hardcoded" en el propio código de la página), ni que el campo de contraseña era el vulnerable (y no el de usuario), ni mucho menos que a alguien se le ocurriera programar una consulta en la que primero aparece el valor y después el campo a comparar (lo cual es perfectamente válido a nivel de sintaxis de cualquier SGBD y nos lo podríamos encontrar por ahí).

Por cierto, si os fijáis en el Hall of Fame, veréis que hay 4 usuarios que sacamos la primera fase prácticamente al mismo tiempo (nomasenviaoeso, samsa, pepeluis y yo mismo). Evidentemente eso no es casual, ya que tras las primeras 3 horas de pruebas desesperantes, en esta ocasión decidimos aliarnos unos cuantos (RoMaNSoFt, Uri, Dreyer y yo) para resolver el reto antes de 24 horas como fuera... ;-)

En medio de esa especie de brainstorming con las ideas de todos nosotros, el primero que dio con la solución correcta fue Uri y a partir de ahí los demás nos fuimos basando en sus pistas para llegar a la misma conclusión.

Nivel 2

Este nivel constaba de varias fases. En la primera de ellas veíamos a Homer sentado sobre un sofá, vistiendo una camiseta roja con un toro negro. En la página no hay ningún control con el que poder interactuar, así que analizamos a fondo toda la página y comprobamos que todo el código que hay sólo sirve para hacer que Homer parpadee de vez en cuando.

Observamos que la página se compone de un mosaico de imágenes y que el nombre del fichero de la imagen central (la correspondiente a Homer) es un tanto aleatorio (todo indica que se ha aporreado el teclado para generarlo), así que intentamos buscar alguna relación con la sesión, con nuestro nombre de usuario, con el navegador usado, etc. Entre las cuatro personas que éramos, la casualidad hizo que al abrir la página desde otro de nuestros navegadores alternativos, Homer apareciera vestido de forma diferente (en esta ocasión con una camiseta del Tío Sam).

En realidad no era cuestión del navegador, sino de las preferencias del idioma, lo cual fue fácil de determinar por el método de prueba y error. Podemos intercalar algún proxy para ir variando los parámetros de la petición y ver lo que sucede (ej: *User-Agent*, *Referer*, etc). Seguramente lo más normal era encontrarse con la imagen de Homer desnudo tras alterar o borrar el contenido del parámetro *Accept-Language*, ya que era la imagen que aparecía por defecto cuando la referencia cultural no era ninguna de las contempladas por los desarrolladores de la aplicación web del reto.

Ahora que ya tenemos la forma de conseguir respuestas diferentes, se tratará de averiguar en qué consiste esta fase. Para ello lo primero que observamos es que todo lo que se envíe después de un guión es ignorado por la aplicación (es decir, obtenemos la misma imagen al establecer las cadenas *es-ES*, *es-US*, *es-AR*, o incluso *es-loquesea*), así que todo lo que vayamos a inyectar tendrá que estar en la parte correspondiente al idioma y no en la de las subetiquetas que identifican la variante del idioma o país.

Si inyectamos alguna comilla simple, romperemos la estructura de la presunta consulta que está ejecutando la aplicación y se nos redirigirá a una página de error. Sin embargo, si inyectamos una cadena que forme una consulta sintácticamente válida, simplemente veremos la imagen de Homer desnudo. Por ejemplo, podríamos probar con estas inyecciones con el fin de determinar el tipo de tecnología utilizada:

```
es'            →  Página de error
e' + 's        →  Página de error
es' or '1'='1  →  Homer desnudo
es' and '1'='1 →  Homer desnudo
es' AND '1'='1 →  Página de error
```

Entre otros detalles, nos llama la atención que la última consulta falle al escribir el operador AND en mayúsculas, lo cual nos lleva a pensar directamente que estamos ante una consulta de XPath, ya que en este lenguaje se tiene en cuenta este detalle (cosa que no ocurre en la sintaxis de SQL, por ejemplo). Ahora se trata de *booleanizar* la inyección, de forma que podamos controlar la imagen que se muestra según la condición que deseemos con el fin de obtener todo el documento XML sobre el que se están ejecutando todas las consultas de XPath. Lo normal hubiese sido que la cuarta consulta de ejemplo (es' and '1'='1) mostrara la imagen del toro, pero no ha sido así...

La razón es bien sencilla y es que nos encontramos ante una consulta un poco más compleja de lo normal, por lo que deberemos utilizar algún paréntesis para ubicar nuestra condición fuera de su ámbito local (para que ésta afecte a toda la consulta). Por ejemplo, comprobamos lo que ocurre al inyectar las siguientes cadenas:

```
es') and (1=1 or 'a'='  →  Homer con la camiseta del toro
es') and (0=1 or 'a'='  →  Homer desnudo
```

Ya tenemos la *booleanización* necesaria para extraer todo el documento XML mediante la técnica descrita por Amit Klein (Blind XPath Injection), de la que ya habíamos hablado en retos anteriores. En el caso de Dani fue relativamente fácil y rápido adaptar la herramienta que ya tenía desarrollada para este menester, de forma que inyectara directamente en el parámetro *Accept-Language*, pero algo falló, ya que la herramienta fue incapaz de obtener el contenido del fichero en primera instancia, debido a que ocasionalmente producía errores en la aplicación de forma sistemática.

Empezó fallando al contar el número de nodos de tipo PI (*Processing Instruction*). Tras omitir ese paso desde el código fuente, se obtenían correctamente el resto de nodos, pero el siguiente fallo apareció al obtener la longitud de las cadenas de texto. Dejando la herramienta de lado y probando con consultas más sencillas de forma manual, Dani se dio cuenta de que las consultas que estaban fallando tenían algo en común: el guión medio...

```
count(/child::processing-instruction())
string-length('abc')
```

Recordemos que la aplicación ignoraba todo lo que había detrás de ese guión, razón por la que estaba fallando la aplicación al quedar la consulta mal estructurada. Teóricamente podíamos haber adaptado la herramienta para que no utilizara ninguna subconsulta que contenga un guión, pero no fue necesario, ya que bastaba con codificar el guión (sustituyéndolo por *%2D*).

Sin embargo, la aplicación volvió a fallar al utilizar cualquiera de las funciones relacionadas con las cadenas de texto (por ejemplo: *substring*, *starts-with*, *concat*, etc). En un primer momento se pensó que se habían filtrado manualmente, pero de nuevo todas ellas tenían algo en común: todas ellas necesitan más de un parámetro y éstos van separados por comas. La solución pasaba por codificar la coma (*%2C*) y ya teníamos la aplicación completamente funcional, por lo que empezó a extraer todo el documento XML de principio a fin.

A pesar de tratarse de una aplicación multihilo y además con alguna que otra optimización, este proceso de extraer carácter por carácter sin búsqueda binaria es lentísimo y su ejecución requirió de varias horas. En cualquier caso, a medida que se iba extrayendo el fichero, se iban desvelando algunos datos ocultos hasta entonces. Por ejemplo, gracias al XML descubrimos que los ficheros CSS estaban en una carpeta llamada “estilosestilosos” o que había una imagen definida para el idioma Xhosa (con el que aparecía un curioso Homer algo tostado, pero no ayudaba a la solución)...

Si analizamos el contenido del fichero XML extraído, se observa la relación de los diferentes idiomas definidos junto con sus respectivas imágenes y hojas de estilo, así como algún que otro comentario más o menos cachondo, la mayoría de ellos para hacernos perder tiempo.

```
<?xml version="1.0" encoding="utf-8"?>
<idiomas>
  <!-- Este fichero contienen las imagenes para las camisetas de Homer... -->
  <idioma lang="es" name="Spanish">
    <!-- Camiseta española -->
    <img>imagenes_l2/__4lo_67_wwwsassdf_juksidkdjfulloka_.png</img>
    <?camiseta file="estilosestilosos/spain.css" ?>
  </idioma>
  <idioma lang="en" name="English">
    <img>imagenes_l2/55_112_ooosldkkfjjs1111XXXxxX___.png</img>
    <!-- En un pueblo italiano   al pie de la montaña    vive nuestro amigo Marco    en
una humilde moradaaa. -->
    <?camiseta file="estilosestilosos/usa.css" ?>
  </idioma>
  <idioma lang="xh" name="Xhosa">
    <img>imagenes_l2/_sd_90992Aspp_lokiksdddes11230____a.png</img>
    <!-- TOP SECRET! peticion_homer_simpson.doc redactada por Way.Smithers-->
    <?camiseta file="estilosestilosos/poralliabajo.css" ?>
  </idioma>
  <idioma lang="nude" name="p0rn!">
    <img>imagenes_l2/nude_homer_12293wfwrsgshdjf_2323232_xxxsdf_.png</img>
    <!--
http://www.informatica64.com/TechNews/link.aspx?id=09ec203fbe930df8a53e17aaa5b022c9 -->
    <?camiseta file="estilosestilosos/nude.css" ?>
  </idioma>
</idiomas>
```

Román resolvió esta parte de forma diferente ya que él no disponía de herramienta de "Blind XPath Injection" propia ni estaba por la labor de crear una. Lo que hizo fue partir de la única herramienta pública existente: BXPI[1]. Su funcionalidad es actualmente muy limitada y de hecho no te permite parametrizar la inyección ni inyectar en una cabecera HTTP cualquiera ni emplear cookies… por lo cual era inútil para el reto… ¿o no? Al menos servía para realizar el proceso de inyección en el caso más básico (un parámetro GET y asumiendo que la consulta XPath es también simple) y en definitiva, con todas las condiciones favorables. La original idea de Román fue encadenar la salida de esta herramienta a un Proxy tipo MITM como "WebScarab" y utilizar las capacidades de scripting en Java de este Proxy ("Bean Shell") para adaptar la inyección básica que implementa BXPI al escenario real del reto (que era totalmente diferente al escenario básico que cubre BXPI).

Veámoslo en detalle:

- **Parametrización de la inyección.**

  BXPI inyectará una cadena como la siguiente:

  ```
  en' or '0'='1
  ```

  Que deberemos transformar, antes de pasarla a la aplicación web vulnerable, en esta otra cadena:

  ```
  en') and ('0'='1' or 'a'='
  ```

  Si os fijáis bien, la transformación deberá consistir en extraer la condición de arriba (en rojo), a la que le falta una comilla de cierre (añadida en verde en la cadena de abajo) y finalmente insertarla en una construcción compatible con el proceso de booleanización que ya explicamos anteriormente.

[1] Blind XPath Injector (BXPI). http://bxpi.codeplex.com/

- **Lugar de la inyección.**

  BXPI no permite inyectar en una cabecera HTTP cualquiera. De hecho, sólo soporta utilizar un parámetro GET. Las peticiones que enviará BXPI serán del tipo:

```
GET
http://retohackingx.elladodelmal.com:80/privado/rhx_level2_homer.aspx?&lang=xxx&
HTTP/1.1
```

  Observamos que BXPI envía incorrectamente separadores de parámetros (“&”) que realmente no son necesarios. Pero en cualquier caso no nos importa: basta con extraer de la petición la cadena de inyección (en rojo). El nombre del parámetro usado tampoco importa y nos lo inventamos nosotros mismos a la hora de configurar BXPI.

  El código Java que se ha diseñado extrae la cadena a partir del primer “=” y que acabe en “&” (que se corresponde exactamente con la cadena marcada en rojo). También se encargará de insertar la cadena convenientemente tratada y modificada en la cabecera “Accept-Language”, es decir, aquella vulnerable a inyección XPath.

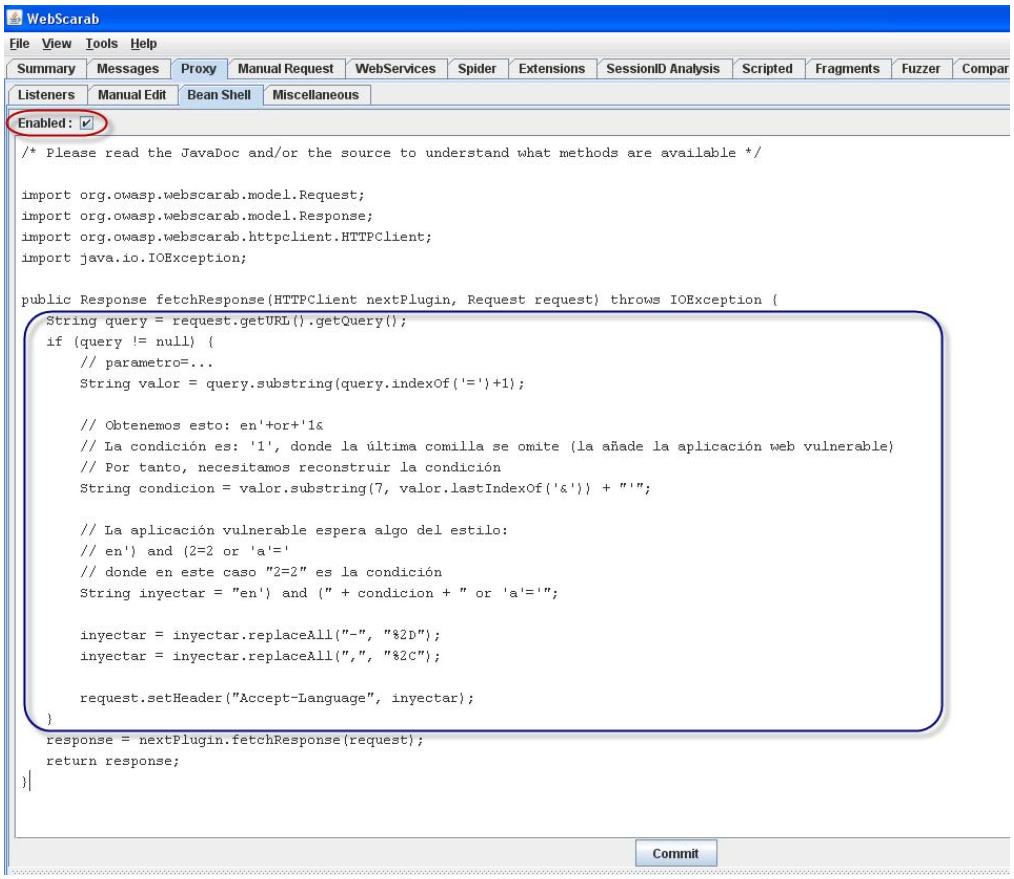
- **Codificaciones especiales.**

  Tal y como vimos anteriormente, el guión (“-”) y la coma (“,”) necesitan ser “escapados”. Para ello los reemplazamos en la cadena a inyectar por “%2D” y “%2C”, respectivamente.

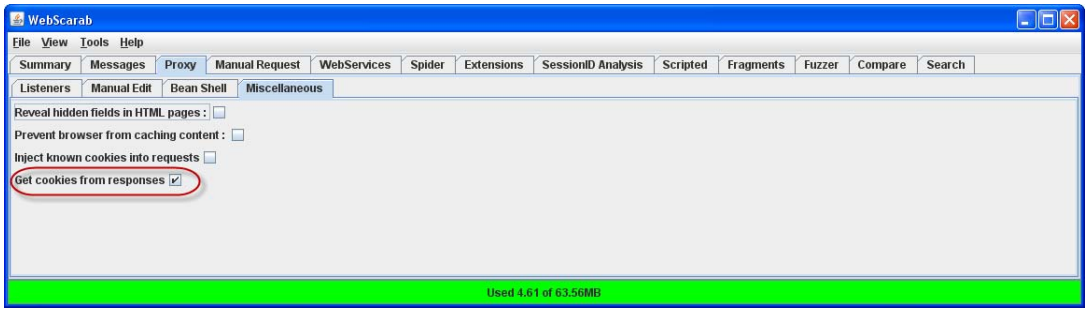
- **Manejo de cookies.**

  BXPI no soporta cookies. Necesitaríamos implementar esta funcionalidad en nuestro script. El problema es que el script no tiene estado sino que es una simple función de transformación que reescribe una petición web realizando siempre los mismos cambios y no tiene una variable de estado que se pueda leer/escribir para pasar datos entre peticiones. Se nos ocurren varias soluciones: una podría ser abrir desde Java un fichero donde se lea/escriba la cookie de sesión, y que antes de lanzar el proceso de inyección habría sido convenientemente inicializado con una cookie válida. Finalmente caemos en la cuenta de que WebScarab ya contempla esta necesidad de manejo de cookies y bastará con marcar un “check-box” en la configuración para tener el problema resuelto.

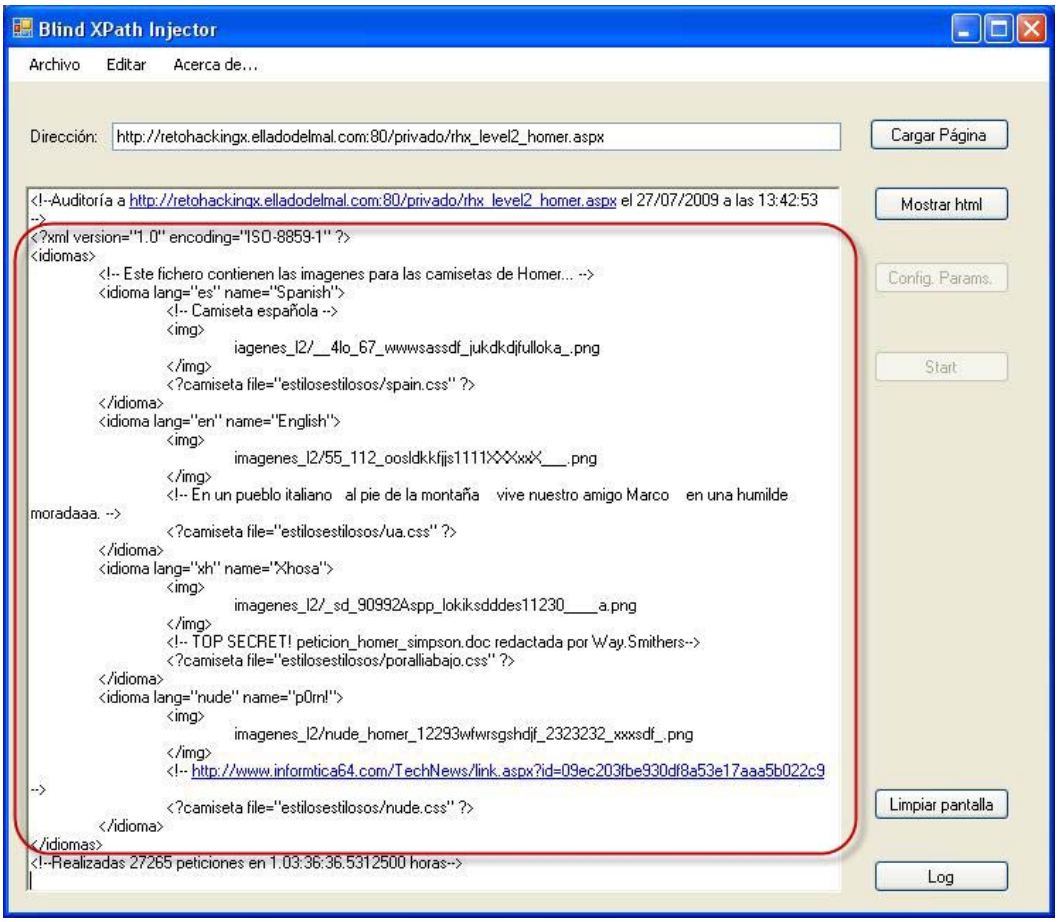
  El script que implementa todo lo explicado más arriba es el siguiente:



Y la opción de WebScarab que permite delegar el manejo de cookies a la propia herramienta se encuentra en:



El proceso completo, una vez configurado WebScarab adecuadamente, pasaría por navegar a la página del reto vulnerable con Internet Explorer con el Proxy configurado (para que WebScarab guarde la cookie de sesión) y posteriormente arrancar BXPI (que tomará la configuración de Proxy del Internet Explorer), teclear la URL (`http://retohackingx.elladodelmal.com:80/privado/rhx_level2_homer.aspx`), pinchar en "Cargar Página" y añadir un parámetro "lang" que se marcará como inyectable. Finalmente arrancamos el proceso de inyección ciega y tras unas horas obtendremos el mismo XML que obtuvo la herramienta de Dani:



Volviendo al XML, todo hacía pensar que la URL del último comentario era la que nos llevaría a la solución de la fase, pero nada más lejos de la realidad, porque en realidad nos redirige a un vídeo de YouTube que nada tenía que ver con el reto (Rick Astley - Never Gonna Give You Up). Puesto que en el contenido de los ficheros CSS tampoco encontramos ninguna información de utilidad, solamente nos queda seguir analizando ese directorio que por su nombre tan peculiar ("estilosestilosos") parece que se nos haya querido ocultar e impedir su acceso fortuito sin haber resuelto esta fase.

Al acceder a la ruta de "estilosestilosos" usando un navegador web, observamos que el servidor nos lista todo el contenido del directorio. Llaman la atención dos fragmentos de ficheros de Word de 162 bytes, ambos idénticos, cuyos nombres eran *"~$temp.doc"* y *"~$tición_homer_simpson.doc"*. Tras examinar minuciosamente todo su contenido, únicamente podíamos leer las cadenas *"Way.Smithers"* y *"Docu"*, por más que intentemos extraer algún metadato adicional (usando la herramienta FOCA online o incluso su versión de escritorio no obteníamos nada).

De nuevo nos encontramos ante uno de esos pasos que normalmente necesitan de muchas pruebas, alguna idea feliz y algo de suerte, porque con los datos que tenemos no está claro cuál es nuestro siguiente objetivo. Si lo pensamos bien, hemos obtenido algún nuevo dato al que aún no le hemos dado uso (el nombre del autor y una parte de una ruta local que seguramente sería la típica *"Documents and settings"*), así que a pesar de no estar seguros de que éste sea el camino correcto, nos centraremos en averiguar la ruta correspondiente a dicho usuario (pero podría haber sido un ataque por diccionario o fuerza bruta al servidor FTP o cualquier otra ocurrencia, quién sabe).

Tras varias pruebas sin éxito basadas en multitud de combinaciones factibles entre el nombre y el apellido (incluyendo solamente las iniciales, con y sin el punto que los separa, etc), junto con palabras que empiecen por "docu" y alguna prueba más, finalmente obtenemos una ruta que nos devuelve un error diferente al 404 (como curiosidad, esta prueba se le ocurrió a Dani recordando el servicio de páginas personales de la universidad y de tantas otras aunque en realidad éstas suelen estar basadas en Unix y no en Windows –que era el caso del reto–):

```
http://retohackingx.elladodelmal.com/~way.smithers
```

Parece que vamos por buen camino, puesto que tirando del hilo hemos dado con la ruta del usuario, pero en esta ocasión el servidor no nos lista el contenido del directorio, así que también tocará averiguar el fichero concreto que buscamos. Por los datos que obtuvimos anteriormente, resulta bastante obvio que el nombre del fichero más probable es *"peticion_homer_simpson.doc"* y así resultó ser.

Lógicamente, se trata de un simple fichero de Word (en su formato binario clásico) y podemos abrirlo para ver que su contenido era el siguiente:

> Estimado trabajador Homer J. Simpson,
>
> El consejo de administración de la central nuclear de Springfield se pone en contacto con Ud. para solicitar su asistencia en una labor rutinaria carente de todo peligro.
>
> Creemos que Ud. es la persona adecuada para realizar la compra de un material de oficina que le facilitara su labor. Para ello, desde su domicilio, use su terminal para navegar por la siguiente URL:
>
> X
>
> Acuda a la sección *Sacapuntas atómicos* y compre el modelo *AtomicSharpener Krusty Edition*. No olvide usar su tarjeta de crédito para realizar la compra.
>
> Al finalizar esta tarea acuda a la cafetería de su centro de trabajo donde tendrá **DONUTS GRATIS** durante toda la jornada.

Seguramente más de uno estará pensando que esa X la hemos reemplazado nosotros para no exponer la solución final, pero no es así. En el documento no aparecía ninguna URL, por más que intentáramos localizar objetos ocultos desde el propio Word, intentando seleccionarlos, activando el control de revisiones, etc. Nuevamente volvemos a intentar extraer algún metadato con la FOCA y en esta ocasión únicamente obtenemos el nombre de dos usuarios (*"Administrator"* y *"Ultimateuser"*), lo que nos podría haber sido de utilidad en algún paso posterior, pero ya os adelantamos que no.

Sin embargo, otro dato que también aparece (incluso en la ficha de propiedades, sin usar la FOCA) es el número de revisiones del documento y en este caso observamos que nuestro documento ha sido modificado y guardado en 5 ocasiones, lo que nos hace sospechar que en alguna de ellas se eliminó algún contenido que seguramente nos pudiera interesar.

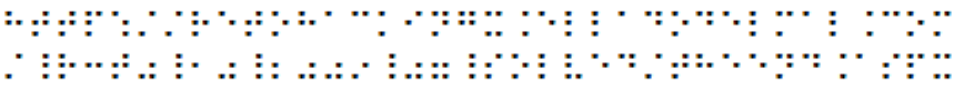
En las versiones relativamente antiguas de Word existía una opción de guardado rápido, la cual aceleraba el guardado de las modificaciones de un documento, a costa de aumentar su tamaño y almacenar los cambios de forma incremental. Esta opción se deshabilitó por completo en el SP3 de Office 2003 (tal y como se indica en este artículo

de la KB de Microsoft), precisamente para evitar situaciones como esta, en la que veremos cómo podremos recuperar una información que había sido eliminada por el usuario que editó y guardó el documento.

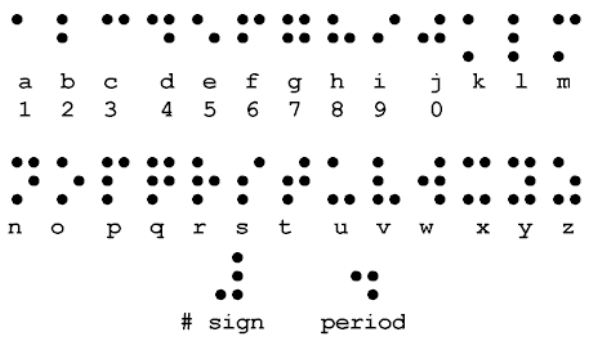
Si abrimos y analizamos el documento con un editor hexadecimal, observaremos que hay una cabecera de un fichero PNG que comienza a partir del byte 5.346 y que termina (IEND) en el byte 37.255. Puesto que en el documento no veíamos ninguna imagen por ningún sitio, parece que el siguiente paso más lógico es el de extraer dicho fragmento y ver qué se esconde en esa imagen. Al hacerlo, vemos una imagen completamente negra de 443 x 50 píxeles, lo que de momento parece bastante inútil.

Seguimos analizando el contenido del fichero y nos encontramos con otra cabecera de otro formato de imagen muy conocido (GIF89a), pero resulta que esa cabecera no aparece en otra parte del documento, sino dentro del PNG que acabamos de extraer hace un momento. Si truncamos la parte inicial de ese fichero hasta llegar a la nueva cabecera y le asignamos la extensión GIF, la imagen es perfectamente visible. Es más, la solución de Román es más simple todavía: cargar el documento Word con un editor hexadecimal y eliminar todos los bytes desde el principio del fichero hasta el comienzo de la cabecera GIF (es decir, justo antes de la cadena “GIF89a”). Parece que a los visores GIF no les importa demasiado que haya “porquería” al final del fichero o al menos al visor que utilizamos, que fue el de Windows.

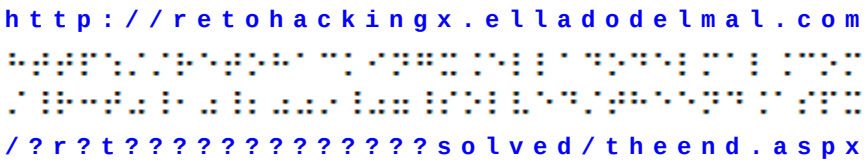
El caso es que al abrir el fichero resultante con un visualizador adecuado (que contemple las animaciones del GIF), veremos una imagen animada (también de 443 x 50 y formada por 38 fotogramas). El último fotograma de la imagen es el que se muestra a continuación (el resto de fotogramas mostraba poco a poco la imagen con un efecto cortina de izquierda a derecha y no tenía más utilidad):



Si la propia imagen no nos dice nada ya de por sí sola, también teníamos una pista a nuestra disposición, ya que el fragmento de la imagen en el fichero binario iba precedido del texto “braile” (lo que probablemente se correspondería con el nombre del fichero original). Efectivamente, se trata de un texto codificado en Braille, el conocidísimo alfabeto de los ciegos, así que para averiguar lo que está escrito no tenemos más que comparar los signos con los de cualquier tabla de referencia de dicho alfabeto, teniendo en cuenta que cada signo está formado por una matriz de tres filas y dos columnas de puntos (presentes o ausentes):



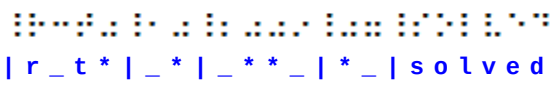
Si utilizamos como referencia cualquiera de las tablas que aparecen en las primeras posiciones de cualquier buscador, veremos que van saliendo la mayoría de los símbolos y es fácil intuir que se trata de una URL, por lo que podemos ir deduciendo algunos de ellos muy fácilmente (como los dos puntos y las barras), pero hay otros que no encontramos por ningún sitio y la cosa se complica un poco:



Todo parece indicar que ya estamos muy cerca del final, pero aún nos falta un buen trozo de la URL y tendremos que encontrar los símbolos que nos faltan, o intentar deducirlos, ya que algunos de ellos se repiten y puede que eso nos permita encontrar algún patrón o pista para intuir alguna palabra.

La verdad es que en el caso de Dani se tuvo la suerte de encontrar una tipografía (fuente o tipo de letra) que contenía exactamente todos los símbolos restantes y ayudándose del mapa de caracteres de Windows los fue sacando cómodamente. Bueno, al decir que tuvo suerte tampoco queremos decir que lo encontrara en cuestión de segundos, porque estuvo un buen rato buscando y no encontró nada útil hasta que pensó que tal vez existiera algún tipo de Braille adaptado al mundo de la informática y buscó la cadena "Unicode Braille" en Google, encontrando entre los primeros 5 resultados una página web con diferentes ficheros TTF de tipografías de Braille descargables desde ahí mismo y probó con la primera de las que aparecían, resultando ser perfectamente válida y adecuada para esta fase del reto.

En cualquier caso, de no haber encontrado la tipografía exacta, podíamos haber aplicado un proceso deductivo, basándonos en las dos letras que habíamos logrado decodificar y en el símbolo de la fila de tres puntos a la derecha, que tenía pinta de ser algún tipo de separador, así como del símbolo que más se repetía (el de los tres puntos representando una especie de esquina inferior derecha).



Si logramos intuir que la primera palabra era "reto", escrita al más puro estilo hacker ("r3t0"), ya tenemos un paso importante, puesto que ahora podremos reemplazar el símbolo del asterisco por ese cero que acabamos de encontrar y tal vez podamos asumir que todos los símbolos restantes fueran dígitos y lanzar un pequeño ataque de fuerza bruta, aunque si seguimos con la racha de contextualizar más aún la URL, podemos intuir que el segundo bloque es un 10 (recordemos que estamos ante el décimo reto) y que el siguiente se corresponde con la fecha (año y mes). De todas formas, aún no hemos dado con el carácter separador, pero eso ya os lo dejamos como ejercicio si os apetece intentarlo... ;-)

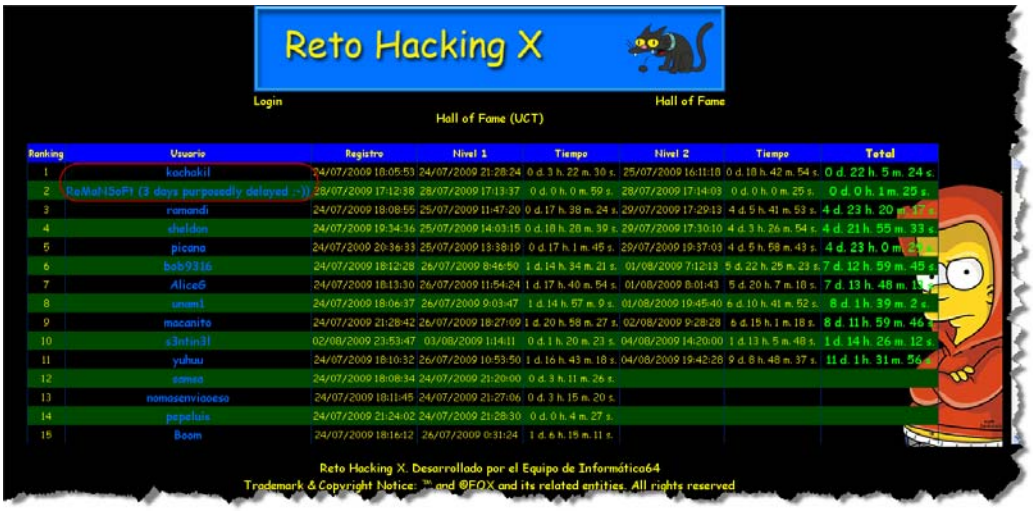
El razonamiento de Román fue similar. Él no tuvo la suerte de encontrar el TTF adecuado pero viendo que faltaban los números y comparando las agrupaciones de puntos que faltaban por resolver con una tabla Braille se podía observar que los puntos

estaban incorrectamente trasladados una fila hacia abajo y que tampoco se había utilizado el símbolo especial para designar números. Deshaciendo el "shift" anterior era trivial obtener los caracteres numéricos restantes y con éstos el separador también era fácil de adivinar.

Tal vez más de uno se extrañe de ver esa representación tan peculiar de los dígitos en Braile, puesto que lo normal (tal y como aparece en la inmensa mayoría de referencias y como estamos acostumbrados a ver en ascensores y otros lugares) sería prefijar con el símbolo de número (una L reflejada horizontalmente) las primeras letras del abecedario (a=1, b=2, c=3, etc). ¿Y por qué no se hizo así? Pues la verdad es que no tenemos ni idea, pero parece que alguno estaba empeñado en complicarnos las cosas con pequeños detalles cuya resolución lleva más tiempo del que parece. O simplemente se equivocaron (bien al traducir a mano o bien por utilizar un .ttf erróneo y arrastrando por tanto los fallos), que es la opción más probable.

En fin, tras este último paso termina el reto (queda resuelto de forma automática al acceder a la URL correcta):

Apareciendo nuestro nombre de usuario en el ansiado *Hall of Fame* y dándonos la satisfacción que ello conlleva.



Como curiosidad para terminar, añadir que ambos finalizamos el reto casi al mismo tiempo (y antes de las 24h del inicio del mismo) pero Román decidió no "firmar" hasta pasados tres días. La razón es que inicialmente él utilizó el XML obtenido con la herramienta de Dani para continuar con el resto de pruebas, puesto que el método del WebScarab era más lento así que posteriormente decidió compensar esa ventaja auto-penalizándose con tres días (que podrían haber sido muchos menos pero de alguna forma se quiso premiar y reconocer el trabajo de los que crearan su propia herramienta de Blind XPath Injection así como darle más emoción al reto).

Agradecimientos y comentarios

Suponemos que a nadie se le habrá escapado el detalle de que por primera vez ambos nos hemos aliado para resolver un reto de Informática 64, así como para redactar este documento. También nos gustaría conocer las impresiones, técnicas y trucos del resto de participantes, ya que no han sido pocos los que han resuelto el reto.

Quién sabe si la próxima vez volveremos a participar juntos o por separado, pero en cualquier caso ha sido una experiencia divertida y motivadora. De no haber sido así, seguramente nos habríamos aburrido mucho antes y no habríamos estado conectados hasta las tantas de la madrugada... ;-)

En fin, queremos terminar este documento agradeciéndole al equipo de I64 por la organización del reto y, sobre todo, al resto de participantes porque sin ellos los retos no tendrían sentido… Y cómo no, a Dreyer y Uri, dos de los más brillantes "Pwndas".

Saludos,

Daniel Kachakil
dani@kachakil.com
Román Medina-Heigl Hernández
roman@rs-labs.com