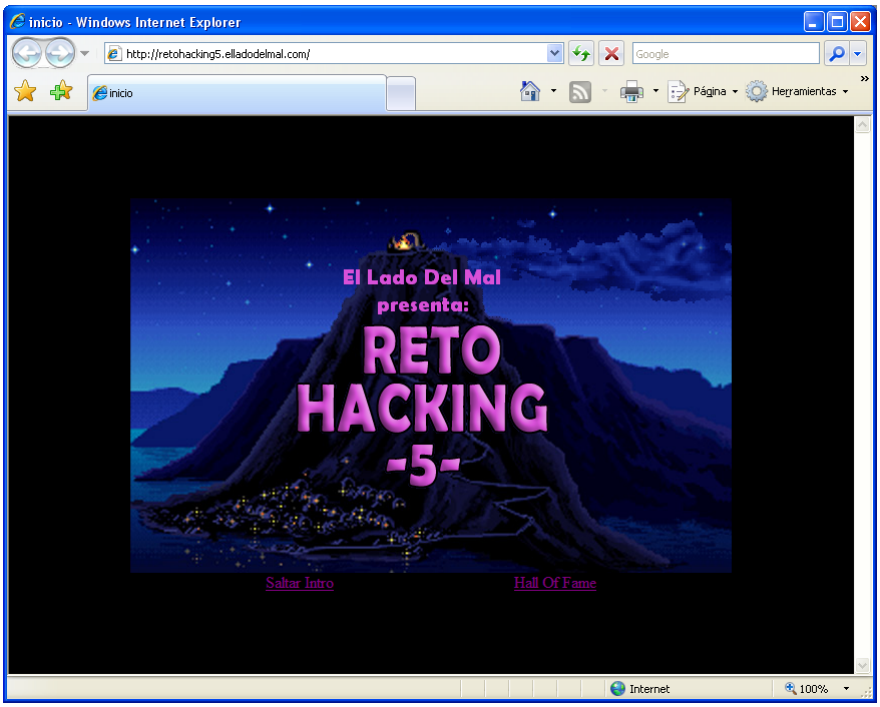
Solución al Reto Hacking V de Informática 64

Enero 2008 © Daniel Kachakil

Introducción

Este documento describe una solución al Reto Hacking V de Informática 64 que se publicó el 14 de diciembre de 2007 en la siguiente dirección web:

`http://retohacking5.elladodelmal.com`



Pistas

Unas horas antes del comienzo del reto se publicaron las pistas en este post. Como comentario personal, si alguien fue capaz de intuir algo útil partiendo de estas pistas, aparte de felicitarle por ello, yo le recomendaría que acudiera a un especialista para que le trate el enredo neuronal que pudiera tener. Otra cosa es que se pueda tratar de justificar el texto de cada pista después de haber dado con la solución, pero no es el caso. Como a mí no me resultaron de ninguna utilidad (más bien al contario, alguna me confundió un poco), mejor vayamos directamente a la resolución del reto.

Parte 1: Las tres preguntas

Tras registrarnos rellenando el formulario de alta y validarnos con los datos de acceso elegidos, accedemos a la primera fase del reto. Lo primero que observamos es que el uso de Adobe Flash es evidente, por lo que en casos como este nunca está de más tener a mano un descompilador (por si acaso, aunque su uso no era imprescindible).

La primera fase consistía en responder correctamente a tres preguntas consecutivas. Viendo el número de respuestas posibles, la probabilidad de pasar esa fase mediante prueba y error era demasiado baja y no parecía el método más razonable, por lo que analizamos el tráfico HTTP que está generando el Flash para detectar posibles vulnerabilidades. Esto lo podemos lograr de varias formas: por ingeniería inversa (examinando el código descompilado del Flash), intercalando un proxy (por ejemplo, Odysseus) o capturando paquetes de nuestra propia interfaz de red (localizando las peticiones GET y POST).

Veamos algunas peticiones realizadas (P*x*) junto con sus respectivas respuestas por parte del servidor (R*x*) y la descripción o comentarios al respecto (C*x*):

P1	**http://retohacking5.elladodelmal.com/Privado/ObtenerDatos.aspx?tipo=numrespuestas&rand=123.45678**
R1	contenido=3
C1	Número que aparece en la frase "Eres el pringao número X que lo intenta"
P2	**http://retohacking5.elladodelmal.com/Privado/ObtenerDatos.aspx?tipo=pregunta&rand=123.4567890**
R2	contenido=¿Para que quiere un pastor un compilador?
C2	Texto de la pregunta actual
P3	**http://retohacking5.elladodelmal.com/Privado/ObtenerDatos.aspx?tipo=respuesta&rand=1234**
R3	contenido=276;Creo que he visto un mono de tres cabezas;El rabino de tu hermana!;El caracol verde;Ha sido él!;Joder, no entiendo nada;La vida es bella, no te hagas preguntas existenciales;La culpa es del gobierno;La casa por el tejado;Me humillo ante ti, ¿y me faltas al respeto?;Me estas vacilando, y tengo unas ganas de darte dos host...;No hablo tu idioma;No;No se si he visto un mono de una cabezas;Pa orejas las de mi suegra;Por el culo te la hinco!;Rumenigue;Sí, claro y también me agacho a por el jabón;Sí;Saca de ahí la cabeza;jaja... muy buena... úñaaao!!!;mmmm... estoy pensando... mmmm;¡3 con las que saques!;¿Y a ti que más te da? ¿Nos fumamos algo?;¿Qué?, no consigo entenderte;¿Donde dices?;¿Eso viene en educación para la ciudadanía?;¿Qué quieres que te diga?, el glamour;¿No lo sabes? ... paleto!;Úbuntu;
C3	Texto de las respuestas posibles, separadas por punto y coma
P4	**http://retohacking5.elladodelmal.com/Privado/ObtenerDatos.aspx?tipo=crespuesta&datos=276&rand=12**
R4	contenido=0
C4	El servidor comprueba si la respuesta a la pregunta es correcta. Devuelve 0 en caso de haber fallado.

Las respuestas mostradas son un ejemplo de un caso concreto. Lo primero que observamos es que cada vez que se realiza una nueva petición, aunque tenga exactamente los mismos parámetros, es muy probable que la respuesta del servidor sea completamente diferente y sin relación aparente con la anterior, debido a que el código del servidor utiliza una secuencia aleatoria independiente de los parámetros. Todas las respuestas parecen ser completamente independientes del parámetro "rand", cuya única utilidad seguramente sea la de evitar los inconvenientes que podrían tener los mecanismos de caché con el Flash ejecutándose en el lado cliente, por lo que podremos prescindir de dicho parámetro en el resto de nuestras pruebas.

Tras realizar un par de pruebas de manipulación de parámetros sin éxito, pensé que si fijaba una de las respuestas posibles y repetía la comprobación de la respuesta realizando peticiones sucesivas, al final terminaría acertando la primera pregunta y el servidor me debería devolver como respuesta "contenido=1" y, efectivamente, así fue. Me extrañó que al volver a repetir dicha petición (P4) una y otra vez, ese valor nunca volvió a valer cero (incluso llegó a valer 2), así que supuse que la vulnerabilidad de esta primera fase tenía que estar basada en haber implementado una funcionalidad importante en el lado cliente, cuando debería haberse implementado en el servidor.

Un programador podría asumir que las llamadas a sus procedimientos se realizarán siempre en el orden en el que las programó, por lo que no sería tan extraño pensar que el fragmento de código que resetea el número de aciertos esté en el método que obtiene las respuestas y que a su vez este método sea invocado siempre que se falle

alguna respuesta. En teoría dicho código sería perfectamente funcional y válido, pero tiene un problema potencial bastante grave. Al igual que ocurre con JavaScript o VBScript, por ejemplo, el código que ejecuta Flash es una secuencia de acciones que puede ser usada para implementar parte de la lógica de la aplicación global. El problema aparece porque, a diferencia de la lógica implementada en la parte del servidor, la que se ejecuta en el lado cliente es susceptible de ser manipulada con relativa facilidad.

A los pocos minutos de haber comenzado el reto y de haber realizado estas pruebas tan sencillas, comprobamos que la respuesta final es "contenido=2", pero aún no sabemos cómo pasar a la siguiente fase. Si descompilamos el Flash, observamos que la última acción que realiza el código es la de redirigirnos de nuevo a la misma página "Nivel1.aspx". Como el servidor almacena la fase en la que estamos de forma permanente, también podríamos haber superado la fase sin más que volver a autentificarnos. Superar esta fase era así de sencillo (repetir P4 unas cuantas veces).

Parte 2: Dolor de espalda

En esta fase aparecemos al lado de una enorme cabeza de mono, en la que se observa una inscripción en uno de sus dientes. Pulsando sobre ese diente nos aparece un texto ininteligible, indicándonos que parece estar en cirílico.

Evidentemente no se trataba de cirílico (porque dicho alfabeto sería distinguible a simple vista), sino de uno de los cifrados más simples que nos podemos encontrar: el cifrado César, que se usaba ya desde hace más de 2.000 años, consistente en desplazar el alfabeto completo N posiciones, es decir, sustituir cada letra por la que está N posiciones a la derecha de la misma. En este tipo de cifrado no vale la pena realizar ningún tipo de criptoanálisis, ya que siempre acabaremos antes probando todas las posibilidades (25, excluyendo la "ñ"). El algoritmo es tan sencillo que no nos costará encontrarlo implementado online en alguna web como esta, en la que descifraremos el mensaje poniendo como clave "D" (es decir, N=3 posiciones).

```
ho pdsd txh wh oohydud d uhvroyhu ho sxcoh vh hqfxhqwud hq od fdushwd
el mapa que te llevara a resolver el puzle se encuentra en la carpeta
uhfxuvrv gh od xqlgdg sulqflsdo
recursos de la unidad principal
```

Ahora que tenemos el texto en claro, comprobamos que se trata de una simple pista y que con esto no es suficiente para pasar a la siguiente fase, por lo que tendremos que seguir buscando. De todas formas, la pista nos lleva a pensar que nuestro objetivo es el de conseguir algún fichero con un mapa en la carpeta "c:\recursos".

Semanas antes de comenzar el reto, estaba leyendo la PCWorld de noviembre cuando me encontré con un artículo, que me resultó bastante curioso, sobre la extracción de ficheros usando técnicas de Blind SQL Injection. La técnica que describía el artículo era tan explícita y fácil de implementar que dudé bastante si esa parte podría aparecer en el reto después de haber sido publicada su solución. El coste de implementar una herramienta para SQL Server 2005 era tan bajo que no dudé en hacerla en su día y comprobar que el método funcionaba perfectamente. El autor colgó en su blog un post con dicha parte del artículo pocos días después de haber comenzado el reto (por cierto, casualmente, el autor del artículo es Chema Alonso)

Habiendo implementado ya la herramienta, el único paso que nos falta para extraer un fichero es localizar una vulnerabilidad de Blind SQL Injection. En la escena de la cabeza del mono aparecen varios objetos que podemos ir pulsando, aunque siempre obtendremos la misma respuesta: "No pienso agacharme a coger ese objeto". A nivel interno, comprobamos que este mensaje se obtiene tras realizar una petición por HTTP como la que se muestra en la siguiente URL:

```
http://retohacking5.elladodelmal.com/privado/obtenerdatospaso2.aspx?idobjeto=2
```

Para comprobar si ese parámetro es vulnerable, simplemente le inyectaremos una condición que sepamos que siempre es verdadera (and 1=1), verificando que la respuesta no varía. A continuación debemos hacer lo mismo con una condición falsa (and 0=1) y comprobar que la respuesta sí varía. En este caso, observamos un tercer comportamiento que nos puede ser de utilidad: si se produce un error en la ejecución de la consulta, el resultado devuelto es diferente de los casos anteriores.

P5	**http://retohacking5.elladodelmal.com/privado/obtenerdatospaso2.aspx?idobjeto=2**
R5	contenido=No pienso agacharme a coger ese objeto
C5	Petición normal
P6	**http://retohacking5.elladodelmal.com/privado/obtenerdatospaso2.aspx?idobjeto=2 and 1=1**
R6	contenido=No pienso agacharme a coger ese objeto
C6	Inyectando una condición verdadera, la respuesta no varía
P7	**http://retohacking5.elladodelmal.com/privado/obtenerdatospaso2.aspx?idobjeto=2 and 0=1**
R7	contenido=No conozco el objeto
C7	Al inyectar una condición falsa, la respuesta ha variado
P8	**http://retohacking5.elladodelmal.com/privado/obtenerdatospaso2.aspx?idobjeto=2xxx**
R8	contenido=
C8	Si se produce un error en la consulta, obtenemos un tercer tipo de respuesta (en blanco)

Una vez localizado el parámetro vulnerable solamente debemos aprovecharlo para aplicar la técnica de extracción, aunque previamente necesitaremos conocer la ruta completa, así como el nombre y la posible extensión del fichero. Para ello podemos inyectar una cadena que verificará que el fichero no esté vacío y que además tengamos acceso al mismo con el usuario actual con el que se esté ejecutando la instancia del motor de base de datos. Teniendo en cuenta la pista que hemos descifrado, iremos probando con diferentes nombres y extensiones de fichero hasta que la siguiente consulta nos devuelva el resultado positivo (R5 o R6):

```
... and (select datalength(c) from openrowset(bulk 'c:\recursos\mapa.jpg',
single_blob) as t(c))>0
```

Ahora que ya tenemos localizado el fichero que buscamos, el último paso de esta fase consistirá en ir obteniendo su contenido byte a byte. Puesto que se trata de un fichero binario, la forma más eficiente de obtener su contenido a ciegas es la de proceder mediante búsqueda binaria para cada uno de los posibles valores de cada byte (de 0 a 255). Para conocer el tamaño del fichero podemos aplicar la misma técnica sobre el resultado de la consulta con la que determinábamos la existencia del fichero, usando la función *datalength*.

A la hora de implementar la herramienta, hemos de tener en cuenta que toda la aplicación web utiliza un mecanismo de autentificación y que el identificador de sesión se almacena en la cookie de ASP.NET llamada .ASPXAUTH, la cual deberemos adjuntar al enviar cada petición. Puesto que el proceso será bastante largo (3675 bytes

que requerirán más de 30.000 peticiones), hemos de tener en cuenta que la sesión podría caducar en cualquier momento o que el servidor podría dejar de responder de forma temporal por cualquier circunstancia, así que no estará de más el ir volcando los datos a disco cuando esto suceda y hacer que la herramienta permita que la descarga del fichero se reinicie partiendo desde la posición en la que quedó. Utilizaremos una inyección como la que se muestra a continuación (sustituyendo los parámetros que se indican entre llaves por los valores oportunos):

```
... and (select substring(c,{posicion_byte},1) from openrowset(bulk
'c:\recursos\mapa.jpg', single_blob) as t(c)) > {valor}
```

Por la extensión del fichero y por su cabecera deducimos que se trata de una imagen en formato JPEG, por lo que tampoco es absolutamente imprescindible obtener el contenido completo del fichero si tenemos un poco de suerte o contamos con alguna herramienta de visualización o edición gráfica más o menos decente, ya que con los primeros 2800 bytes ya podemos visualizar lo que se esconde en esa imagen, es decir, la URL que tendremos que teclear para acceder a la tercera fase. Omitiendo los 5 primeros caracteres de la página, esta era la puerta que nos llevará a la última fase:

```
http://retohacking5.elladodelmal.com/privado/?????fase_3r3tohacking5.aspx
```

Parte 3: Robando a CaraLimon

En esta ocasión aparecemos al lado de una choza y de una cabeza que esconde otro texto en "cirílico" que descifraremos utilizando el mismo tipo de cifrado y el mismo desplazamiento que la fase anterior.

```
ho qrpeuh gho ilfkhur hv pdsduhsolfdhqflulolfrbuhgxflgr.eps
el nombre del fichero es mapareplicaencirilicoyreducido.bmp
```

Parece que de nuevo la cosa va de obtener ficheros, pero esta vez no será necesario pasar por el servidor de base de datos como hicimos con la fase anterior (reconozco que fue lo primero que estuve intentando un buen rato sin éxito). Se trataba de aplicar el principio KISS, es decir, ir directamente a la solución más simple, sin complicarse la vida innecesariamente. Solo teníamos que descargar el fichero directamente desde el servidor web, metiendo ese nombre en la URL del navegador.

```
http://retohacking5.elladodelmal.com/privado/mapareplicaencirilicoyreducido.bmp
```

La imagen que obtenemos tiene unas dimensiones bastante reducidas (50x32 píxeles), tal y como se indica en su nombre. Al abrirla vemos un dibujo de un supuesto mapa con un interrogante y una serie de puntos grises bastante dispersos. Como ya no tenemos más pistas a las que agarrarnos, todo apunta a que la imagen esconde algo que no vemos por más que la ampliemos. Se trata de un mensaje oculto de alguna forma.

En el mundo de la esteganografía podemos encontrar infinidad de técnicas de ocultación de información, no solo en imágenes sino en casi cualquier tipo de medio que nos podamos encontrar (audio, vídeo, etc). Es más, podemos encontrar muchísimos ejemplos que nada tienen que ver con la informática, debido a que estas técnicas se empezaron a aplicar desde hace unos cuantos milenios.

En el caso que nos ocupa, partiremos de la hipótesis de que existe un mensaje oculto en el fichero BMP. Si analizamos la estructura del fichero observamos que comienza por una cabecera de 54 bytes que define su formato (tamaño, profundidad de color, etc), a continuación le siguen 1020 bytes que definen la paleta de colores (4 bytes por cada uno de los 255 colores que contiene) y por último nos encontramos con la información gráfica, en la que cada píxel se representa con un byte (cuyo valor se usa como índice que apunta al color que se encuentra en esa posición de la paleta).

Aunque en general se podría haber ocultado cualquier tipo de fichero en formato binario, existen dos tipos de información que serían más fácilmente detectables dentro de un fichero BMP: otra imagen y texto plano.

En general, la técnica más común para ocultar información consiste en aprovechar los N bits de menor peso de cada byte del fichero (excluyendo la cabecera), de forma que se podrían utilizar uno o varios bits de cada byte. Evidentemente, cuantos más bits se utilicen para ocultar información, mayor será la diferencia con la imagen original. Tras aplicar un filtro de plano de bits para cada uno de los 8 posibles bits, descartamos la opción de que se trate de una imagen oculta dentro de otra, ya que la información se habría revelado visualmente de forma inmediata. Por tanto, parece que todo apunta a que se trata de un texto oculto (probablemente en formato ASCII).

Por empezar por el caso más simple, asumiremos la hipótesis de que solamente se utiliza uno de los bits y que este bit es el de menor peso. De esta forma, por cada 8 píxeles del fichero se consigue ocultar un byte de información, variando la imagen de una forma prácticamente imperceptible.

Llegados a este punto conviene detenernos un poco y reflexionar sobre nuestras suposiciones, ya que estamos asumiendo como ciertas demasiadas hipótesis al mismo tiempo. Hasta ahora no tenemos ningún indicio real de que la imagen contenga información oculta. En caso de tenerla, no sabemos si es un texto y tampoco podemos garantizar que utiliza un único bit de cada byte. En caso de ser así, tampoco podríamos asegurar que se tratara del bit de menor peso. No sabemos si el mensaje estará en formato ASCII. ¿Y si el algoritmo utiliza solo los bits de los bytes pares, o un bit de cada tres bytes, o cualquier otra implementación perfectamente válida que se nos pudiera ocurrir? A lo mejor la solución no es tan fácil como podríamos pensar.

Parece que lo más razonable en este momento es aplicar un algoritmo de fuerza bruta que vaya probando la mayoría de las combinaciones posibles, ya que en el momento en que falle cualquiera de las hipótesis, no obtendremos ningún tipo de información válida. Para no detallar innecesariamente cada una de las pruebas que implementé, vamos a resumirlo en que fui metiendo un bucle dentro de otro, juntando los bits en grupos de 8 para formar bytes. Por otro lado, le aplicaba una función que sumaba un punto por cada byte que coincidiera con el valor ASCII de cualquier letra minúscula (97-122), mayúscula (65-90), espacio (32) o dígito (48-57). Los casos de prueba que obtenían más puntos, contenían más caracteres del alfabeto y son los que revisaba de forma manual para ver si había algo legible.

Debido a que el nombre del fichero hacía referencia al cirílico, tampoco sería descabellado pensar que además de estar oculto, el mensaje estuviera también cifrado con el mismo algoritmo que aparecía en dos de las tres fases (cifrado César). De

cualquier forma, estaríamos buscando valores dentro del mismo alfabeto, por lo que el método descrito funcionaría igualmente.

A pesar de todas las posibilidades que contemplé, al principio no obtuve ningún texto legible en ninguno de los casos, porque no tuve en cuenta un factor que era crucial: para que la cosa funcionara, simplemente teníamos que recorrer el fichero en sentido inverso.

Para obtener el mensaje oculto tendremos que leer los bytes del fichero empezando por el final, tomando el bit de menor peso de cada byte, agrupándolos de 8 en 8 habiendo ignorando previamente los 2 primeros bits de la secuencia. Con esto obtenemos un vector de bytes de salida que forma una cadena de texto legible en la que de nuevo omito los últimos caracteres de la contraseña (y así os animo a intentarlo):

```
http://retohacking5.elladodelmal.com/privado/?????Fase_3R3toHacking5.aspx
The_Secret_Of_Monkey_Island_Graphic_Aventure.??????????????????
By Alfonso Muñoz. (http://stegsecret.sourceforge.net)
```

El paso final consiste en acceder a la URL de la tercera fase que ya conocíamos e introducir la contraseña en el cuadro de texto que aparece tras pulsar sobre la puerta de la choza. ¡Al fin hemos conseguido colar nuestro nombre en la lista de ganadores!

Volviendo a la parte 1: la idea original

Sí, ya hemos superado el reto, pero parece ser que la primera fase la hemos superado de una manera inesperada. Resulta que el fallo de programación que habíamos aprovechado para pasar esa fase fue algo accidental y no había sido contemplado por los creadores del reto. Evidentemente ya era demasiado tarde como para arreglarlo sin poner a nadie en clara desventaja, así que se dejó tal cual.

Mientras algunos hacían sus apuestas con cervezas para ver quién resultaba ganador del reto, yo hacía mis apuestas particulares sobre las tecnologías y vulnerabilidades que podrían usarse para montar un reto. El primer posible indicio me lo guardé de la charla de RoMaNSoFt (4-10-07, Getafe), en la que nos relataba a los allí presentes un poco de historia sobre la vulnerabilidad de Blind XPath Injection que aparecía en el tercer reto, comentando que Amit Klein publicó un documento sobre ese tema en 2004, pero sin haber liberado la herramienta que aseguraba tener implementada como prueba de concepto (y que tampoco le hizo llegar a Román a pesar de la petición expresa que le hizo por e-mail).

Para los que os perdisteis aquél evento y para refrescar la memoria a los que estuvisteis, aquí tenéis un enlace con el vídeo de la charla de Román (la parte a la que me refiero aparece en el minuto 12:30 y en la diapositiva 14).

Por otro lado, podemos encontrar otro pequeño indicio sobre ese mismo tema en este post, publicado además en la revista PCWorld de julio:

> *En la versión 4 de la herramienta se añadieron módulos para la detección de vulnerabilidades Blind SQL Injection y Blind XPath Injection (de estas últimas podríamos hablar algún día por esta sección).*

El tercer indicio lo encontré en la charla que de Chema Alonso en el Technology Tour de Valencia, simplemente porque volvió a mencionar el tema aunque muy por encima, pero le noté bastante seguridad al afirmar públicamente que se podía extraer un fichero XML aprovechando esa vulnerabilidad.

Así que después de tantos indicios en la misma dirección, me puse a recopilar información sobre Blind XPath Injection y no encontré nada especialmente útil, aparte de la ya mencionada publicación de Amit Klein. No obstante, en dicho documento se explica una técnica de extracción del XML subyacente usando dicha vulnerabilidad, así que solo era cuestión de implementarlo.

Para ello me construí una pequeña aplicación web vulnerable a Blind XPath Injection y otra aplicación cuyo objetivo era el de extraer ese XML a ciegas, siguiendo los pasos que se explicaban en el documento escrito por Amit Klein y completando la información con la documentación del W3C sobre XPath y XML. A pesar de que el proceso no se explica completamente y que el documento contiene varios errores (yo creo que intencionados), finalmente conseguí el objetivo con mi aplicación de prueba. El siguiente paso fue probarlo en la primera parte del Reto Hacking 3, en la que existía una vulnerabilidad de XPath Injection, comprobando que también funcionaba.

El procedimiento genérico completo consiste en obtener el número de nodos que hay de cada tipo (elemento, texto, comentario o instrucciones de procesamiento). Hecho esto, vamos recorriendo los nodos por orden de posición y determinamos el tipo al que pertenece cada uno de ellos. Si resulta ser de tipo elemento, entonces volvemos a aplicar el método de forma recursiva, extrayendo sus posibles atributos y nodos internos. Si es de tipo texto o comentario, simplemente vamos extrayendo su contenido letra a letra. Como se puede observar, no nos hace falta ningún tipo de información previa sobre la estructura del XML, que podría ser todo lo compleja y heterogénea que queramos.

Sin saber si me serviría para el reto, por lo menos ya tenía una herramienta que era capaz de obtener absolutamente todo el XML, desde los elementos y atributos hasta los comentarios e instrucciones de procesamiento que pudiera contener. De momento parece que esa vulnerabilidad no se ha utilizado en este reto… ¿o tal vez sí?

Tras haber superado el reto, Chema me confirmó que la idea de la primera fase era precisamente la de extraer el XML de preguntas y respuestas aprovechando una vulnerabilidad de Blind XPath Injection. Sabiendo que yo tenía desarrollada una herramienta para ello, me pidió que intentara superar el reto de esa forma y así lo hice, aunque no resultó tan fácil detectar el parámetro vulnerable.

Recordemos la petición que se utilizaba para comprobar si una respuesta era correcta o no (la que denominaba como P4):

```
http://retohacking5.elladodelmal.com/Privado/ObtenerDatos.aspx?tipo=crespuesta&datos=276
```

Intentamos localizar cual era el parámetro vulnerable, pero parece que ninguno varía su respuesta con el estado inicial, aunque el parámetro "datos" parece el candidato que tiene más papeletas. Para conseguir dos tipos de respuestas diferentes teníamos que acertar la primera de las preguntas a base de repetir la petición fijando una respuesta válida, de forma que el servidor nos devolviera "contenido=1". Solo a partir de ese

momento comprobamos que las siguientes peticiones devuelven resultados diferentes de forma permanente (mientras dure la sesión):

P9	**http://retohacking5.elladodelmal.com/privado/ObtenerDatos.aspx?tipo=crespuesta&datos=' or (1=1) or 'a'='b**
R9	contenido=1
C9	Inyectando una condición verdadera, la respuesta no varía
P10	**http://retohacking5.elladodelmal.com/privado/ObtenerDatos.aspx?tipo=crespuesta&datos=' or (1=0) or 'a'='b**
R10	contenido=0
C10	Al inyectar una condición falsa comprobamos que la respuesta ha variado

Ahora que tenemos el mecanismo inicializado, basta con aplicar la técnica de extracción, configurando la herramienta para que envíe la cookie de sesión, etc. Una vez terminado el proceso obtenemos el fichero XML subyacente, cuyo contenido parcial es el que se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<RetoHacking5>
  <Preguntas>
    <Pregunta id="1">
      <cas>¿Deberas depilarte tu gran CEJA?</cas>
    </Pregunta>
    <Pregunta id="2">
      <cas>!Bonitas orejas¡, ¿has intentado planear?</cas>
    </Pregunta>
    <!-- En vez de este comentario, aquí va el resto de preguntas -->
    <Pregunta id="26">
      <cas>¿Quiéres usa el comodín del público?</cas>
    </Pregunta>
  </Preguntas>
  <Respuestas>
    <Respuesta id="1">
      <cas>Creo que he visto un mono de tres cabezas</cas>
    </Respuesta>
    <Respuesta id="2">
      <cas>No se si he visto un mono de una cabezas</cas>
    </Respuesta>
    <!-- En vez de este comentario, aquí va el resto de respuestas -->
    <Respuesta id="30">
      <cas>mmmm... estoy pensando... mmmm</cas>
    </Respuesta>
  </Respuestas>
</RetoHacking5>
```

Curiosamente, hay 26 preguntas y 30 respuestas listadas sin relación aparente. Si las intentamos relacionar directamente por el identificador no conseguimos nada. Parece increíble, pero aún nos queda un pequeño paso más y es aquí cuando entra en juego una variable a la que aún no le habíamos dado utilidad: el número de "pringao" es el que relaciona las preguntas con las respuestas, así que tendremos que recordarlo para toda la tanda de respuestas. Por cierto, debido a otro pequeño fallo (probablemente relacionado con la caché), el número de "pringao" que se muestra tras haber cometido un fallo es erróneo, ya que la segunda vez aparece el mismo valor que en la primera ocasión (cuando ese valor no corresponde con la respuesta que nos devuelve el servidor) y la tercera petición muestra el valor que tuvo que mostrarse en la segunda, etc.

Localizando el identificador de la pregunta que aparece, sumándole el número de "pringao" y obteniendo el resto de la división entre el total de respuestas (módulo 30), obtenemos el identificador de la respuesta correcta a esa pregunta. Ahora sí que superamos la primera fase como estaba previsto. ¿A que no era tan difícil? ;-)

Agradecimientos y comentarios

Esta vez hay que reconocer que el reto estaba muy bien ambientado desde el primer momento en el que se publicó la introducción que anunciaba la fecha y hora de publicación del reto. La música, los gráficos y unas animaciones impecables que recreaban algunos escenarios del Monkey Island estaban muy currados, así que lo primero es felicitar a Rodol y a Álex por su excelente trabajo.

En cuanto a la parte técnica, siempre aprendemos algo nuevo gracias a las ideas de Chema y esta vez no ha sido menos, aunque lástima por el fallo de la primera fase. Como crítica constructiva y por el bien de todos, esperemos que la próxima vez no se aplique la ley de Parkinson ("El trabajo crece hasta llenar el tiempo de que se dispone para su realización") y que el siguiente reto salga a la hora prevista y sin fallos.

Por último, gracias a ti que estás leyendo este texto. Espero no haberte aburrido mucho y sobre todo espero que te haya sido de utilidad. ¡Anímate a intentarlo!

Saludos,

Daniel Kachakil
dani@kachakil.com