

Solución al Reto Hacking VI de Informática 64

Febrero 2008 © Daniel Kachakil

Introducción

Este documento describe una solución al Reto Hacking VI de Informática 64, el primero de la segunda temporada, que se publicó el 1 de febrero de 2008 en la siguiente dirección web:

<http://retohacking6.elladodelmal.com>



Pistas

Esta vez las pistas prometían ser útiles... Se publicaron unas horas antes del reto, adelantando la existencia de dos partes y diciendo lo siguiente:

Primera parte

Sé que cuando lleguéis a esta parte vais a estar un poco... descolacados. Como un piloto de Formula 1 con el coche cruzado en la salida. Ya sabéis lo que mi primo el nano piensa sobre qué se debe hacer para ganar, ya que nunca es algo hecho. Si fuera fácil no habría que hacer tanto trabajo. Sí, sé que el reto será fácil, pero ganar nunca.

Una vez lleguéis allí, debéis saber que yo me siento agradecido por todos los que os habéis sumado a esta edición del reto. Sé que restar tiempo a vuestras horas de ocio, trabajo y en definitiva vida dividiéndoos entre la familia y amigos

y el Reto Hacking es un gran trabajo, pero espero que con este reto se multiplique un poco vuestro conocimiento sobre lo ya conocido con una forma nueva de trabajo.

Segunda parte

Para la segunda parte... la cosa está de partes otra vez. Deberéis agudizar la Vista en este examen de reflejos. El que tenga más reflejos lo sacará antes....

Parte 1: Un poco de práctica para aprobar el examen

Para acceder a la primera fase, lo primero que teníamos que hacer era registrarnos eligiendo nuestro nombre de usuario e introduciendo una dirección de correo válida, donde recibiremos la contraseña correspondiente generada aleatoriamente. Accediendo con estos datos en la página de login, observamos que cambia el menú de navegación, apareciendo tres nuevos elementos:

- **Práctica:** En este apartado encontramos un formulario con un campo de texto y un botón de enviar que nos invita a practicar con el álgebra de Garrupito, preguntándonos cual es el valor de $1000+1$.
- **Examen:** Aquí encontramos un formulario similar al anterior, preguntándonos por el valor de Garrupito.
- **Subir nota:** Para poder subir nota, primero tendremos que aprobar el examen. (Esta sección corresponde a la segunda fase y se habilitará cuando tengamos aprobado el examen).

Tras echarle un vistazo a las secciones disponibles, decidimos empezar por el principio, practicando un poco. La primera prueba necesariamente tenía que ser 1001 , por ser la más lógica y evidente, pero no conseguimos nada, al igual que tampoco sirve de nada interpretar el valor como binario, o el operador de suma como concatenación y otras pruebas de ese estilo. Probamos con diferentes cadenas de texto aleatorio, comillas simples y dobles, operadores lógicos (and, or, not, xor), matemáticos (+, -, *, /, ^), de manejo de cadenas (len, trim, rtrim) y comprobamos que todas ellas son procesadas con normalidad. De momento, podemos asegurar las siguientes observaciones:

- Al introducir un valor numérico o expresión booleana (ya sea verdadera o falsa), obtenemos como respuesta *"Vamos, vamos, garrupintizate!"* [R1]
- Al introducir cualquier cadena o expresión errónea, obtenemos como respuesta *"ACCESS: Error al convertir valor numérico"* [R2]

Llegados a este punto, Chema publica un post con una pista adicional (más bien insistiendo en la pista existente), lo que nos lleva a pensar que no vamos por buen camino, ya que si no interpretamos la pista adecuadamente, parece ser que no lograremos nada. Retomando la primera pista, deducimos inmediatamente que se trata del piloto de Fórmula 1 Fernando Alonso y según la nueva pista, tendremos que buscar unas declaraciones que nos indicarán el camino a seguir para ganar. Tras haber leído decenas de declaraciones del piloto, me quedé con unas que decían que había que dar el máximo, otras que hablaban de vueltas y otras en las que decía que había que ir siempre al límite, pero no sabía qué hacer con ello.

De todas formas, para mí fue fundamental el tercer post ("*O resolvéis el misterio de la primera pista o vais a tener que hacer algo más de dos millones de peticiones algo así como....*"). Directamente me vino a la cabeza el valor máximo que se puede almacenar en un entero de 32 bits con signo (al que llamaremos MAXINT), que no son dos millones, sino más de dos mil millones: 2147483647 ($2^{31} - 1$). Al introducir este valor, obtenemos una nueva respuesta: "*Uy que cerca, casi estás garrupintizado!*" [**R3**]

Esto me llevó a pensar que iba bien, que había que buscar un límite pero que no era exactamente ese valor, así que fui probando inútilmente diferentes valores en busca de ese otro límite inexistente. En ese momento me di cuenta de que el mensaje de error contenía una pista clave (la palabra "ACCESS") y eso parecía indicar que ahí detrás existía un motor de bases de datos Access (Microsoft Jet) procesando nuestras peticiones, por lo que intentaremos encontrar la forma de verificar la existencia de este motor de base de datos, buscando algún mecanismo que nos diferencia dos tipos de respuesta para los valores verdadero y falso (proceso de "booleanización"), realizando posteriormente las subconsultas que nos interesen sobre ese mecanismo.

Como primer paso, ya vimos que obteníamos tres posibles respuestas según el tipo de entrada (valor numérico [**R1**], valor no numérico [**R2**] o MAXINT [**R3**]), por lo que únicamente nos falta relacionar un par de ellas con una subconsulta adecuada. Si hacemos caso a la pista, parece que debemos usar las respuestas **R1** y **R3**, de forma que nuestra subconsulta devuelva un valor MAXINT si es verdadera y cualquier otro valor si es falsa (o viceversa). Por ejemplo, introduciendo la siguiente subconsulta:

```
2147483647 * (select max(1) from tabla)
```

La mayoría de funciones de agregado (MAX, MIN, AVG, etc) devuelven el valor nulo si no tienen datos de entrada (es decir, si la consulta no devuelve registros). Las operaciones matemáticas que involucran al valor nulo no provocan ningún error y devuelven también el valor nulo. Sin embargo, la función COUNT devolvería cero en ese caso. Para la consulta anterior, si la tabla tiene algún registro, la subconsulta devolverá 1, que multiplicado por el valor MAXINT lógicamente dejará intacto este valor y obtendremos la respuesta **R3**. Por el contrario, si la tabla no tiene registros, entonces la subconsulta devolverá el valor nulo, que multiplicado por MAXINT también devuelve el valor nulo y obtendremos **R1**. Si la tabla no existe, la consulta devolverá un error (**R2**), por lo que la existencia de una tabla se puede determinar fácilmente, simplemente ejecutando una consulta de este tipo. Una consulta equivalente a la anterior podría ser esta (en la que seleccionamos un solo registro, con valor 1):

```
2147483647 * (select top 1 1 from tabla)
```

Sobre este tipo de subconsultas también se podrían haber hecho otras variaciones que consiguen el mismo resultado, como utilizar una suma o una resta en vez de la multiplicación. Por ejemplo:

```
2147483646 + (select max(1) from tabla)
2147483648 - (select max(1) from tabla)
```

También existía otra posibilidad, que tal vez resultara más evidente y natural, ya que en este caso no necesitamos usar la pista, ni llegar a ningún tipo de valor especial. El método se basa en que una división por cero devuelve un resultado que no está definido como numérico (por lo que nos devolverá el mensaje del error **R2**), mientras

que una división por cualquier otro valor no nulo, devolverá un número (es decir, como respuesta obtendremos el mensaje **R1**). Si la tabla no existe, o no tiene registros, esta consulta fallará:

```
1 / (select count(*) from tabla)
```

Otro tipo de variación de este método es el de provocar un desbordamiento, multiplicando un valor muy alto por cero (obteniendo **R1** como respuesta) o por un valor mayor que uno (obteniendo **R2**). Por ejemplo:

```
(2^1023) * (select max(2) from tabla)
```

Habiendo concluido el reto, Chema me comentó que la consulta que ejecutaba el servidor estaba basada en la conversión al tipo numérico a través de la función de valor absoluto (ABS), lo cual me llevó a deducir que otro tipo de inyección también podría ser válido, siempre que no exista alguna mecanismo que escape o duplique los paréntesis (cosa que tampoco se hacía). Por tanto, esta consulta también funcionaría, siendo tal vez la más flexible de todas ellas:

```
0) or (select count(*) from tabla)
```

Como última opción, también podríamos haber recurrido a las [consultas pesadas](#), en las que no importa el mensaje que se nos devuelva, sino el tiempo que tarde el procesamiento de nuestra petición. De todas formas, este método es demasiado invasivo y no me parece razonable su uso en un reto ni en otro tipo de servidores, ya que el retardo se consigue consumiendo muchos recursos del servidor, teniendo como efecto colateral que todas las peticiones restantes (las de los demás participantes y usuarios en general) se demoren mientras se ejecuta nuestra consulta pesada. En el momento en que se lanza una consulta mientras se está ejecutando una consulta pesada, nunca sabremos cual de las dos es la que ha generado el retardo y el método deja de ser válido. Una consulta pesada no es comparable a ejecutar instrucciones del tipo *waitfor* o *sleep* (que en Access no tienen su equivalente), ya que éstas han sido diseñadas específicamente para dejar pasar el tiempo de espera indicado, pero sin consumir gran cantidad de recursos del servidor, por lo que su uso sería perfectamente válido.

Ahora que tenemos tantas posibilidades a nuestro alcance, vamos a resolver esta fase utilizando la que se suponía que teníamos que utilizar, es decir, la que hace uso del valor MAXINT, pero antes que nada nos aseguraremos de la existencia del motor de base de datos de Access. Sabemos que Access tiene algunas tablas de sistema que son accesibles por defecto. Dependiendo de la versión, nos podremos encontrar con las siguientes tablas: **MSysAccessObjects** (versiones 97 y 2000), **MSysAccessStorage** (versiones 2003 y 2007). Comprobamos que en nuestro caso tenemos éxito con la segunda prueba y de momento podemos suponer que vamos por buen camino:

```
2147483647 * (select max(1) from MSysAccessObjects) → R2 (Error)
2147483647 * (select max(1) from MSysAccessStorage) → R3 (Verdadero)
```

Recordemos que el examen consiste en introducir el valor correcto de Garrupito, por lo que todo apunta a que dicho valor se encuentra en la base de datos a la que tenemos acceso a través de la técnica de Blind SQL Injection descrita. Las bases de

datos de Access no tienen catálogo accesible, por lo que tendremos que adivinar el nombre de la tabla y de sus campos. Sería razonable ejecutar este tipo de consultas:

```
2147483647 * (select max(1) from garrupito) → R3 (Verdadero)
2147483647 * (select count(*) from garrupito) → R3 (Verdadero)
2147483647 * (select max(1) from garrupito where garrupito > '') → R2 (Error)
2147483647 * (select max(1) from garrupito where valor > '') → R3 (Verdadero)
```

Ya tenemos la tabla (garrupito), el número de registros (1), el campo que nos interesa (valor) y su tipo (texto), así que ahora podemos aplicar esa técnica que ya conocemos, a la que hacía referencia la primera pista. Primero obtenemos la longitud del valor o número de caracteres, como siempre (búsqueda binaria):

```
2147483647 * (select max(1) from garrupito where len(valor) > 64) → R1 (F)
2147483647 * (select max(1) from garrupito where len(valor) > 32) → R3 (V)
2147483647 * (select max(1) from garrupito where len(valor) > 48) → R1 (F)
2147483647 * (select max(1) from garrupito where len(valor) > 40) → R1 (F)
2147483647 * (select max(1) from garrupito where len(valor) > 36) → R1 (F)
2147483647 * (select max(1) from garrupito where len(valor) > 34) → R1 (F)
2147483647 * (select max(1) from garrupito where len(valor) > 35) → R3 (V)
2147483647 * (select max(1) from garrupito where len(valor) = 36) → R3 (V)
```

Aunque la última consulta realmente es innecesaria, si hacemos el procedimiento a mano nunca está de más una comprobación adicional por si nos hemos equivocado en algún punto. Por último, iremos obteniendo el valor de cada carácter utilizando el mismo método, actuando sobre los valores ASCII. Siempre que se pueda, es preferible evitar una consulta con el operador LIKE, ya que éste no distingue entre las mayúsculas y las minúsculas.

```
2147483647 * (select max(1) from garrupito where asc(mid(valor,{i},1)) > {Xi})
```

Una vez hayamos obtenido los 36 valores ASCII los reconvertimos de nuevo en caracteres para formar la cadena que buscamos. Este es el valor de Garrupito (como siempre, os dejo la parte final para que terminéis los deberes en casa):

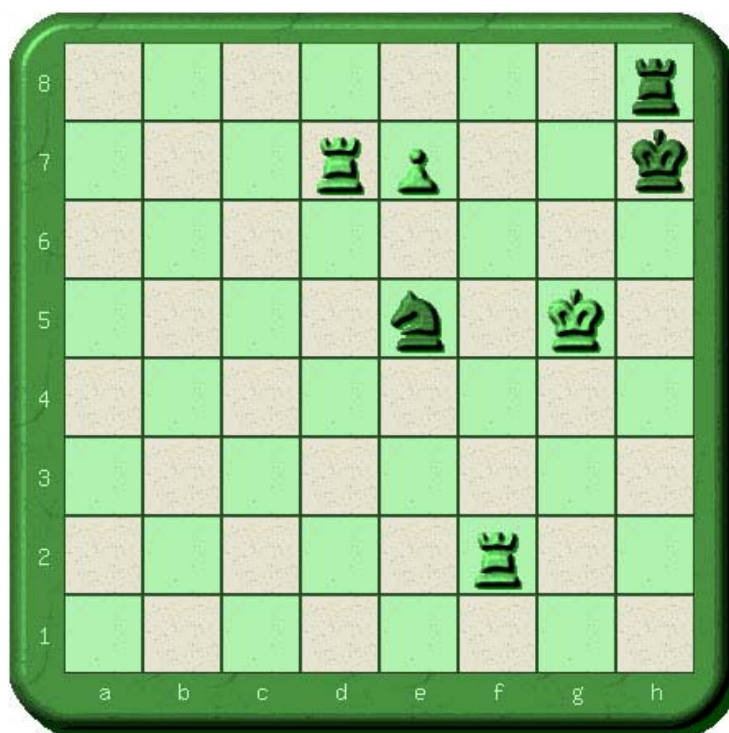
```
9DB7D157-CD68-42a3-A366-C32BE377????
```

A pesar de su formato hexadecimal de 32 caracteres (que podría ser una hash MD5, por ejemplo), antes de intentar hacer algo raro con esa cadena vamos a probar lo más sencillo, es decir, introducirla directamente como respuesta a la pregunta del examen. Finalmente comprobamos que la solución era así de simple, teniendo en cuenta que teníamos que introducir la cadena exacta, respetando las mayúsculas (y cómo no, respetando también esa "a" minúscula que había por ahí). ;-)

Como comentario adicional, Palako descubrió que el valor MAXINT estaba almacenado en un registro de una tabla llamada "valores", en el campo "valor", lo cual también podría ser aprovechado para construir consultas algo más complejas.

Parte 2: Subir nota

Como no queremos conformarnos con ese miserable cuarto de punto que obtenemos al superar la primera fase, vamos a intentar subir nota superando también la segunda y última fase del reto. Al acceder a esta parte, nos encontraremos con un ejercicio curioso para hacer en casa. Nos descargamos un fichero PNG y tenemos que responder a la pregunta de cómo se puede hacer mate en un solo movimiento si jugamos con las blancas.



Cualquiera que conozca las reglas del ajedrez sería capaz de determinar rápidamente que con esa disposición de piezas no hay forma de hacer mate en una sola jugada, sino que hacen falta dos: adelantar el peón a E8 (promocionando a dama) y moviendo la torre de F2 a F7.

Parece que la pregunta tiene truco, así que intentamos introducir diferentes movimientos (incluso aunque sean inválidos) y otras cadenas de texto buscando alguna vulnerabilidad sin éxito. La pista decía que teníamos que agudizar la vista, que la cosa va de partes otra vez y el enunciado de la propia fase nos indica que el ejercicio lo tenemos que trabajar en casa. ¿Será otra prueba de esteganografía?...

Hay un pequeño detalle que me llamó la atención y es que a pesar de que el formato PNG no tiene pérdida de calidad, la imagen del tablero no era del todo nítida, como si estuviera comprimida en JPEG. Parece que la elección del formato no ha sido casual, pero no vemos nada especial en la imagen y tampoco en la metainformación que podría contener.

Vamos a documentarnos un poco sobre el [formato PNG](#), ya que conociendo su estructura interna tal vez encontremos algo útil. Es curioso que el formato defina una etiqueta específica para marcar el fin de la imagen (IEND) y es más extraño aún encontrar tanta información después de esa etiqueta. Este hecho podemos comprobarlo incluso con la funcionalidad de búsqueda del bloc de notas.

```

,?>Q(0,0)ÉjS-f%â»tIt+P'»"ÚeTfnÚ@2âf:†C00
8$'00†G0>g900000°§0ñ00/±ú€0, L0°03€0%âA00âè|pÿzú$0"p01A4vP@.T)€>ÉP-wizf_`x$8}kvhsI%âfC
00 8$p} e.%YÁE[ÖU\ -kwo-00-00`0# 0f->.MÉÿ0ay0é`4$0 IEND0B',Rar!00 I"se 0
iZs#0#0é00'F0bÿ0I0É0â5"
60dÉâ0jÁ00 ,FizwYIH"Q0'BS0..v#z«1ÿ,x,0|0c¶Ág#`ix0Q0+)Tñ_z0)YzS-*0úâ`"Nf0}}0-Á]_+f)0i80
`Cz0A" P07-ÿH00 â+«0#Á 0EGÜ0A000,0u€00
§0#)=I`aé0âP40?0é00`0#S-S0%|0600áúI0..20G$&"0{çv00"0#0xÁ?3$ ,0Mÿ0Wj+v-+000&0>0. t00b=/ $
é00içA0y0%00b1Á'0>"0`0k00v-|bÿN!00k00Éb%Q0$0ÿ,±0;0«0j$[.d'
Á00Éx0q%0H6"».(ed$"8$0S\j06000€ç€>0AJO`0ÿ€#00k`Rfäüßi#0'èè10E1¶ÁÈ'"<±0ñ0>*2÷iZ0(0"çA
p00É0Wl!0v']`éi00ri€);`i00âj,0¶7i}§B"qum'pI0;0zm ]00
ÿñ0#tEÿ'†f090A00b<800«;0æCN0i0i0G05_N0ÉÉ"çp]0z
0.#$A`Ü#z0"0000A±`w0|?'wgIi€Jÿy>r0EÉ0'Á€)>§00«0N0"ÉwEij0ñq09Q0PRA9ÁL000*j0<000<0`Éñçÿ
0É04x0`Z0i`Ü004$1`Á0>Aæ00má$ f,0ñp0ÿi$âv000 008]i&M0'z<Ák1000TELÿN±fju0ié00æ$2k0i00æ
pm000?mÁ0 ä"mè;0ÿGW>0>0`0»--n"00ÿJ0[e10iN000?0-0ÿ%|?ú& |3âiq-d0
$wí00i-É0m'zæ80`j4ú-00%æ0ZNê+0v
z_d(y000z0Añ0"ÿ00A"t0JZ0à0A;¿â0I'†0ÿu0vÉhsá0kr¿\«0""p>2i1èµI000

```

Si utilizamos un editor hexadecimal y eliminamos todo el contenido que aparece después de la etiqueta IEND, comprobamos que el nuevo fichero generado se visualiza exactamente como estaba, así que deducimos que esa parte realmente no pertenece a la imagen y que a su vez es ignorada por el código que está renderizando el PNG. Por otro lado, entre tanto texto ilegible también llama la atención encontrarnos con el texto "Rar!" justo después del final de la imagen. Caracteres que casualmente coinciden con los de la cabecera de cualquier fichero comprimido con RAR. En realidad esta técnica se puede considerar como aplicación de esteganografía, ya que logra el objetivo de esconder un mensaje (en forma de fichero) dentro de otro.

Ahora solamente nos queda extraer esa parte interesante (desde "Rar!" hasta el final del fichero), guardándola como nuevo fichero con la extensión ".rar", volcando el contenido con algún editor hexadecimal. También se podría haber renombrado el PNG directamente sin modificarlo, aunque no sé si esto dependerá de la implementación del descompresor que usemos (como mínimo parece funcionar bien con el WinRAR v3.50)

Sin embargo, al abrir el fichero comprimido nos encontramos con un último paso, ya que se encuentra protegido por contraseña, pero probando con "garrupito" conseguimos abrirlo (de no haber sido tan trivial, tendríamos que haberlo hecho con fuerza bruta). Dentro del RAR encontramos otro fichero PNG en el que está la solución que ya conocíamos: más o menos dice que es imposible resolverlo en un solo paso. El reto termina introduciendo la frase exacta (incluyendo algún error ortográfico que aparecía por ahí).

Agradecimientos y comentarios

En esta ocasión el reto se publicó a la hora prevista (20:00) con una puntualidad increíble, lo cual se agradece. También me ha gustado la idea de mostrar la tabla de concursantes, ya que contenía información interesante como la hora de registro de cada uno, la evolución por fases en tiempo real, el tiempo invertido en cada una de ellas, etc.

Esperemos que la próxima vez las pistas sean un poco menos rebuscadas y más útiles (tal vez la insistencia en profundizar en la pista de Alonso era innecesaria). Tampoco estaría de más que no se permita el acceso a los ensamblados de la aplicación ASP.NET ni a los ficheros de log, que siempre hay algún copión suelto por ahí, aunque reconozco que la técnica usada para ello es muy buena y no se me habría ocurrido... ;-)

Gracias a los que han hecho posible este reto (Chema, Rodol, Álex y Filemaster). Gracias a las ideas y puntualizaciones de RoMaNSoFt, Mandingo y Palako, que han enriquecido este documento. Gracias a SealTeam por hacernos pasar un rato divertido jugando un poco con alguna DLL de este reto y del anterior. En general, gracias a todos los que habéis participado en este reto y, como siempre, gracias a ti que estás leyendo esto, esperando que te haya sido de utilidad.

Saludos,

Daniel Kachakil

dani@kachakil.com