

Solución al Reto Hacking VIII de Informática 64

Julio 2008 © Daniel Kachakil

Introducción

Este documento describe una solución al Reto Hacking VIII de Informática 64, el tercero de la segunda temporada, que se publicó el 4 de julio de 2008 en la siguiente dirección web:

<http://retohacking8.elladodelmal.com>



Pistas

Supongo que el planteamiento del reto estaba bastante claro desde el principio, por lo que no se publicaron pistas previas al inicio del mismo.

Fase 1: Análisis forense de un fichero ".pcap"

Como viene siendo habitual en los últimos retos, para acceder a la primera fase teníamos que registrarnos con un nombre de usuario y una dirección de correo válida, donde recibiremos la contraseña correspondiente generada aleatoriamente. Una vez registrados y autenticados, accedemos a la fase 1.

En esta ocasión no teníamos que encontrar ningún tipo de vulnerabilidad, ya que por primera vez el reto no iba de eso, sino que tendremos que demostrar nuestras habilidades detectivescas en un reto de análisis forense. Todavía no tenemos muy claro nuestro objetivo, pero sí el primer paso, que era tan sencillo como descargarse un fichero comprimido que contenía otro llamado *Trama.pcap*

Por si alguien se enganchó en ese mismo punto, si no conocemos la extensión del fichero nunca está de más hacer una búsqueda previa:

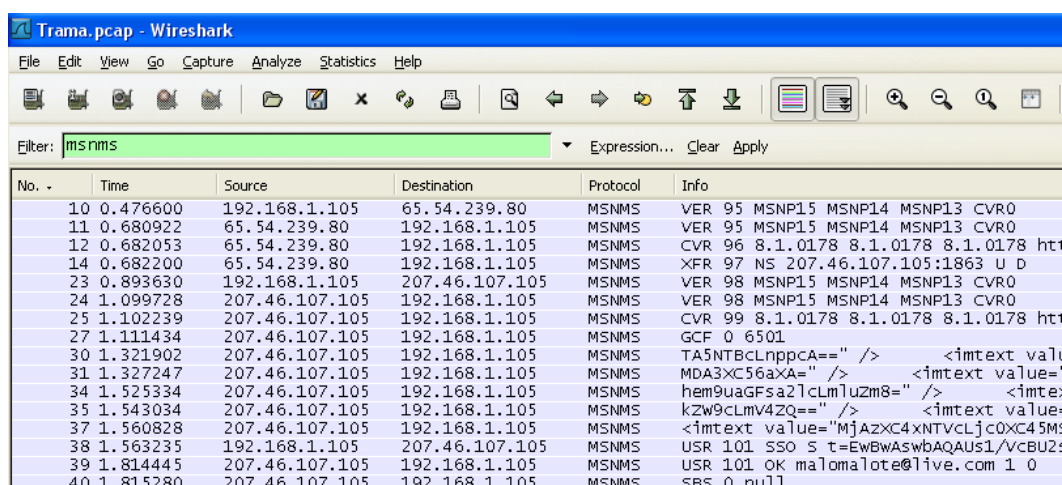
<http://www.fileinfo.net/extension/pcap>

<http://es.wikipedia.org/wiki/Pcap>

Rápidamente vemos que se trata de un fichero que contiene capturas de paquetes de una red (obtenidas mediante un sniffer). Un excelente programa para abrir este tipo de fichero es el [Wireshark](#) (anteriormente conocido como [Ethereal](#)).

Una vez descargado e instalado, basta con abrir el fichero con este programa para visualizar toda la secuencia de paquetes que ha sido interceptada y almacenada. Vemos que hay todo tipo de secuencias de diferentes protocolos clásicos como DNS, NetBIOS, ARP, ICMP, HTTP, UDP, TCP, etc.

Llama la atención que desde el principio del fichero nos encontremos con tantos paquetes marcados como *MSNMS*, es decir, del protocolo de MSN Messenger. Si aplicamos un filtro para visualizar únicamente este protocolo (tecleando *msnms* en el cuadro de texto *Filter* de la ventana principal), tal vez podamos leer la conversación y enterarnos de alguna pista.



No. -	Time	Source	Destination	Protocol	Info
10	0.476600	192.168.1.105	65.54.239.80	MSNMS	VER 95 MSNP15 MSNP14 MSNP13 CVR0
11	0.680922	65.54.239.80	192.168.1.105	MSNMS	VER 95 MSNP15 MSNP14 MSNP13 CVR0
12	0.682053	65.54.239.80	192.168.1.105	MSNMS	CVR 96 8.1.0178 8.1.0178 8.1.0178 ht
14	0.682200	65.54.239.80	192.168.1.105	MSNMS	XFR 97 NS 207.46.107.105:1863 U D
23	0.893630	192.168.1.105	207.46.107.105	MSNMS	VER 98 MSNP15 MSNP14 MSNP13 CVR0
24	1.099728	207.46.107.105	192.168.1.105	MSNMS	VER 98 MSNP15 MSNP14 MSNP13 CVR0
25	1.102239	207.46.107.105	192.168.1.105	MSNMS	CVR 99 8.1.0178 8.1.0178 8.1.0178 ht
27	1.111434	207.46.107.105	192.168.1.105	MSNMS	GCF 0 6501
30	1.321902	207.46.107.105	192.168.1.105	MSNMS	TA5NTBcLnppcA="" /> <imtext valu
31	1.327247	207.46.107.105	192.168.1.105	MSNMS	MDA3xC56axA="" /> <imtext value='
34	1.525334	207.46.107.105	192.168.1.105	MSNMS	hem9uaGFsa2lclmluZm8="" /> <imte:
35	1.543034	207.46.107.105	192.168.1.105	MSNMS	kZw9cLmV4ZQ="" /> <imtext value:
37	1.560828	207.46.107.105	192.168.1.105	MSNMS	<imtext value="MjAzXC4xNTVcljc0XC45M!
38	1.563235	192.168.1.105	207.46.107.105	MSNMS	USR 101 sso s t=EwBwAswBAQAUS1/vcBU2:
39	1.814445	207.46.107.105	192.168.1.105	MSNMS	USR 101 OK malomalote@live.com 1 0
40	1.815280	207.46.107.105	192.168.1.105	MSNMS	SRS 0 null

Fig. 1 – Visualizando el fichero en Wireshark con un filtro aplicado

Buscando más información sobre este protocolo, nos encontramos con que es propietario de Microsoft y no está publicado, por lo que todo lo que aparece en los siguientes enlaces, no es oficial y es fácil que no esté actualizado, que a veces no esté del todo completo o incluso que tal vez sea incorrecto:

<http://www.hypothetic.org/docs/msn/index.php>
<http://msnpiki.msnfanatic.com/index.php>

De momento no hace falta entender todo el protocolo que utiliza este programa, ya que si inspeccionamos a ojo el contenido de los paquetes podemos observar las direcciones de correo electrónico correspondientes a cada usuario y leer la conversación que aparece como texto plano en los paquetes *MSG*. Era la siguiente:

Nº	De	Mensaje
814	MaloMalisimo	buenas
832	MaloMalote	hola
838	MaloMalisimo	te envio los posibles puntos de encuentro y lo que me pediste
846	MaloMalote	ok
...		
2839	MaloMalisimo	te dejo que creo que me siguen
2861	MaloMalote	ok, suerte!

Es evidente que el texto de la conversación no da la solución a la fase, pero ya nos aporta una pista, puesto que habla de un envío de "algo". Si analizamos más a fondo el contenido de otros paquetes, observamos que el inmediatamente posterior al 846 (es decir, el 854, asumiendo que la vista sigue filtrada para visualizar solamente el protocolo MSNMS) contiene el siguiente texto codificado en Base-64, concretamente en el parámetro *Context*:

`fgIAAAMAAAi1gsAAAAAAAAAABQAHQAbwBzAEQAZQBFAG4AYwB1AGUAbgB0AHIAbwAuAHAAbgBnAA...`

Decodificando dicho texto comprobamos que la parte final corresponde con el texto claro *PtosDeEncuentro.png*, por lo que parece evidente que ese "algo" que se enviaba era un fichero PNG. Si nos entretenemos decodificando el resto de mensajes en Base-64 que se envían por el mismo protocolo, encontraremos un encabezado de PNG, e incluso podremos recomponer el fichero completo, pero eso solo es la miniatura que se envía incluso antes de aceptar y comenzar la recepción del fichero.

El fichero real se envía directamente del emisor al receptor (P2P), sin utilizar los servidores de MSN que actúan como intermediarios en las conversaciones normales. Por tanto, quitamos el filtro en el Wireshark y visualizamos de nuevo toda la trama de paquetes para darnos cuenta de que existe un tráfico bastante importante por TCP/IP entre las direcciones 192.168.1.105 y 192.168.1.107. Desde esta última IP se inicia el SYN en el paquete 985, recibe el ACK en el 986 y queda claro que a partir de ahí existe una secuencia o stream TCP que parece tener su interés para superar el reto.

Para recomponer la secuencia basta con pulsar con el botón derecho sobre cualquier paquete de la misma y seleccionar del menú contextual (o del menú *Analyze*) la opción *Follow TCP Stream*. Entonces nos aparecerá una ventana con toda la secuencia TCP/IP entre ambas direcciones. Sin embargo, tras echarle un vistazo general

a toda la secuencia, determinamos que solo nos interesa el tráfico de un único sentido, ya que es el que contiene el fichero propiamente dicho. Por tanto, seleccionamos del cuadro de lista desplegable la opción que muestra solamente los paquetes enviados desde 192.168.1.107 hacia 192.168.1.105 y exportaremos el contenido a un fichero. Para ello es imprescindible seleccionar la opción adecuada (*RAW*), ya que de otra forma el fichero resultante no se corresponderá con la secuencia original.

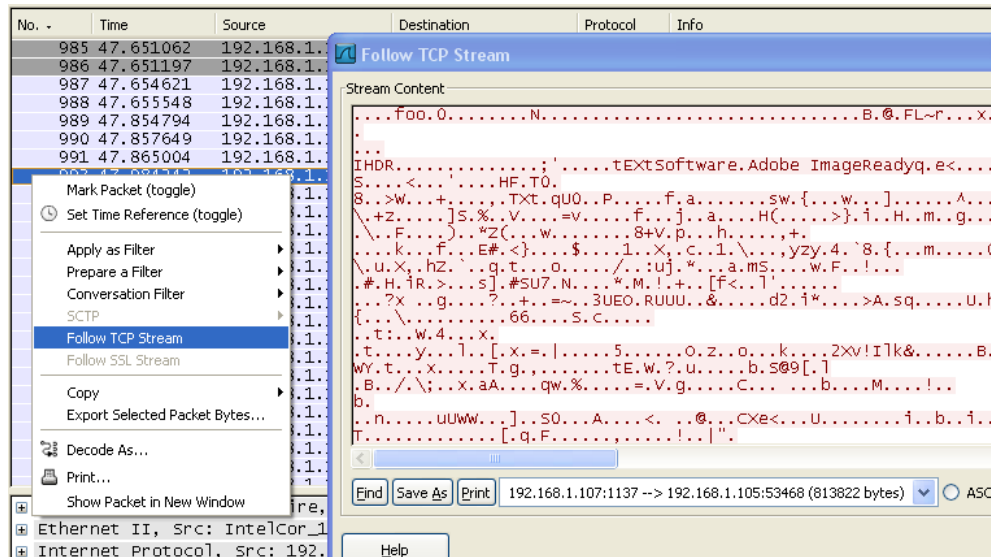


Fig. 2 – Opción "Follow TCP Stream" en Wireshark

Una vez exportado el fichero, comprobamos que la cabecera del PNG aparece después de 112 bytes, pero incluso eliminando esa parte, el resultado no parece corresponder con un fichero PNG válido. Revisando la secuencia con un editor hexadecimal comprobamos que cada cierta distancia aparecen algunos bloques de datos sospechosos, ya que no parecen corresponder al fichero. Analizando este tipo de bloques llegamos a la conclusión inicial de que tienen una longitud de 52 bytes, que comienzan por "78 05 00 00" (hex) y aparecen a intervalos regulares de 1404 bytes, así que optamos por eliminarlas pero aún así no se visualiza el PNG. Más adelante veremos que esto no es del todo cierto...

Está claro que nos hemos dejado algo por el camino y es que con todo lo que hemos hecho, no era tan difícil toparse con un encabezado "PK" y con un *Cliente.exe* que aparecía por ahí muy cerca. La clave del fallo es que hemos asumido erróneamente de que se trataba del envío de un fichero PNG, cuando en realidad se trata de un envío simultáneo de dos ficheros, tal y como podemos comprobar si además del anterior (854) analizamos también el contenido del paquete 940 (de nuevo el *Context* que aparece en Base-64), ya que contiene el texto *Cliente.zip* y eso nos llevará a adoptar otra metodología de trabajo orientada a diferenciar y extraer los bloques de cada fichero por separado.

Ahora es cuando entran en juego esas cabeceras de 52 bytes que antes habíamos eliminado tan alegremente, ya que contienen información muy valiosa que no podemos ignorar (incluso en el caso de haberse transferido un único fichero, este método no habría funcionado). Si volcamos todos esos bytes y analizamos minuciosamente todas

las cabeceras que comienzan por "78 05 00 00" y que habíamos asumido que tenían una longitud total de 1404 bytes, nos damos cuenta de que la mayoría de bytes son cero o son idénticos en todas ellas, pero otros varían, tal y como podemos apreciar en esta tabla cuyos valores están todos en decimal:

Offset general	Tamaño	Bytes (posición relativa)								Fichero
		1	2	5	9	13	14	21	22	
60	1404	120	5	185	211	0	0	34	214	PNG
1464	1404	120	5	187	212	0	0	3	25	ZIP
2868	1404	120	5	185	211	72	5	34	214	PNG
4272	1404	120	5	187	212	72	5	3	25	ZIP
5676	1404	120	5	185	211	144	10	34	214	PNG
7080	1404	120	5	187	212	144	10	3	25	ZIP
8484	1404	120	5	185	211	216	15	34	214	PNG
9888	1404	120	5	185	211	32	21	34	214	PNG
11292	1404	120	5	187	212	216	15	3	25	ZIP
12696	1404	120	5	185	211	104	26	34	214	PNG
14100	2451	120	5	185	211	176	31	34	214	PNG
16551	1404	120	5	185	211	248	36	34	214	PNG
17955	1404	120	5	185	211	64	42	34	214	PNG

Por las cabeceras, sabemos que el primer paquete corresponde al fichero PNG y el segundo al ZIP, por lo que no resulta complicado concluir que los bytes 5, 9, 21 y 22 tienen una relación directa con el fichero. Puede que indiquen su tamaño total, o que sean parte de algún identificador, aunque tampoco nos importa demasiado para lograr nuestro objetivo, que no es otro que separar ambos ficheros. Sin embargo, si juntamos las partes correspondientes al ZIP, no conseguimos recomponer el fichero, ya que no se podrá descomprimir correctamente con ningún descompresor.

Por otro lado, en la tabla hemos calculado el tamaño del paquete restando el offset actual del siguiente, pero comprobamos que existe un salto inesperado que resalta otro error del planteamiento ya que aparentemente tenemos un paquete de tamaño 2451, cuando en realidad en ese bloque se esconden dos paquetes. No se trata de localizar la secuencia en hexadecimal "78 05 00 00", sino de leerla e interpretarla correctamente. Esos 4 bytes nos están indicando la longitud del resto del mensaje, en orden inverso (en nuestro caso, 0578 en hexadecimal, o lo que es lo mismo, 1400 bytes).

Por ello debemos corregir la información anterior e interpretar correctamente los paquetes correspondientes a ese bloque, ya que todo parece indicar que nos falta el último trozo del fichero ZIP.

Offset general	Tamaño	Bytes (posición relativa)								Fichero
		1	2	5	9	13	14	21	22	
14100	1404	120	5	185	211	176	31	34	214	PNG
15504	1047	19	4	187	212	32	21	3	25	ZIP
16551	1404	120	5	185	211	248	36	34	214	PNG

Ahora que ya hemos conseguido separar ambos ficheros, descomprimos el ZIP, extrayendo y ejecutando el fichero *Cliente.exe* (que requiere .NET Framework 2.0), comprobando que se trata de una sencilla aplicación que nos solicita una contraseña para obtener una supuesta clave de cifrado.

Para analizar su funcionamiento interno nada mejor que descompilarla usando alguna herramienta como [.NET Reflector](#) y localizar el código que se ejecuta al pulsar el botón *btRecuperar* del formulario principal (llamado *Clave*). Vemos que hay una instrucción IF que comprueba si la contraseña introducida tiene una longitud de 9 caracteres y además hay un bucle que verifica que dicha contraseña se corresponde con una almacenada en un vector de enteros incluido en el mismo método. Si la contraseña coincide, entonces se realiza una llamada a un servicio web.

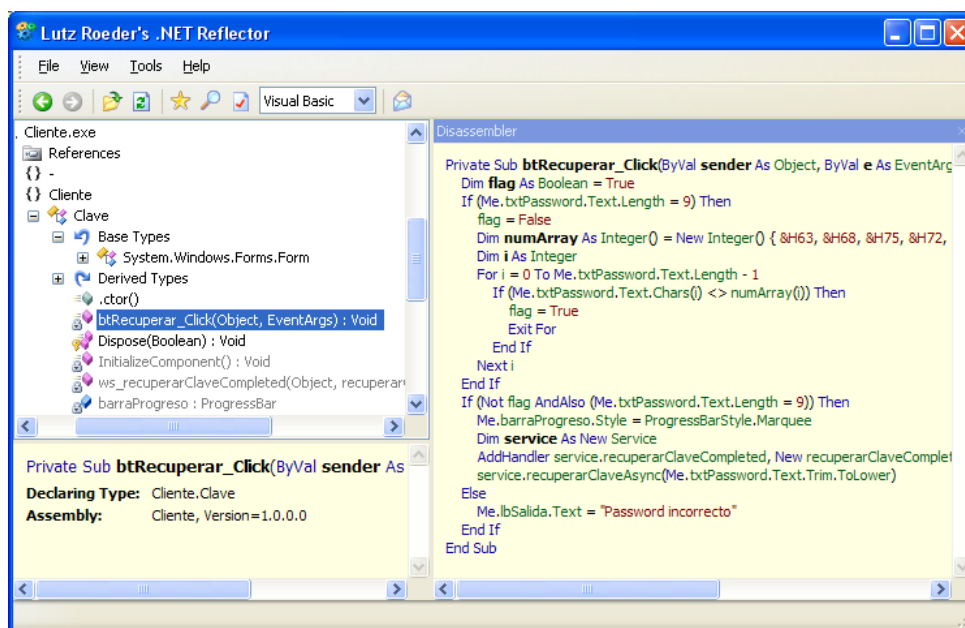


Fig. 3 – Código descompilado usando .NET Reflector

El código es tan evidente que no vale la pena elegir un camino que no sea el de determinar la correspondencia directa de los caracteres de la contraseña en ASCII (es decir, *chur******) e introducirla en el programa en ejecución para que continúe ejecutando el resto del código, obteniendo así la palabra clave de cifrado necesaria para superar la fase 1, es decir, *[[^*****^]]*. Por cierto, el fichero PNG no tenía ninguna utilidad para superar el reto, pero por simple curiosidad, este era su contenido:



Fig. 4 – Fichero PtosDeEncuentro.png (reducido)

Fase 2: Análisis forense de un disco duro

Tras introducir la clave de la fase anterior, accedemos a la segunda y última fase del reto, en la que se nos piden cuatro datos para superarlo (sujeto, sitio, día y hora) y también se nos indica los pasos a seguir para descargarnos un fichero RAR (de más de 400 MB, por cierto). Dicho fichero comprimido contiene un único fichero llamado *XPForensics Hard Disk.vhd* y por la extensión del mismo podemos deducir que se trata de una imagen o disco duro virtual en formato Virtual Hard Disk.

Se trata de un formato patentado por Microsoft, usado principalmente por las herramientas Virtual PC y Virtual Server, aunque la [especificación del formato VHD](#) es pública y parece bastante simple, por lo que hay varias herramientas que la soportan. Si disponemos de cualquiera de estas herramientas, podemos intentar arrancar la máquina virtual instalada, pero en principio no lograremos acceso al sistema operativo, ya que los usuarios del mismo están protegidos por contraseña.

De todas formas, tampoco es necesario arrancar el sistema operativo, ya que para tener acceso a su estructura de ficheros bastará con incluir la imagen como disco duro secundario de cualquier otro sistema operativo. Para ello podremos utilizar por ejemplo el propio Virtual PC sobre otra máquina virtual que tengamos instalada, o bien podemos montar la unidad en nuestro propio sistema operativo si nos instalamos el [Virtual Server 2005 R2 SP1](#) (no nos valdrían versiones anteriores) y seguimos las indicaciones que aparecen en [este post](#) de David Cervigón.

Como aún no sabemos lo que buscamos, podemos empezar explorando un poco todos los directorios típicos en busca de algún fichero que nos dé alguna pista. Por ejemplo, el escritorio, la carpeta *Mis documentos*, directorios temporales, ficheros típicos como los DBX de Outlook Express, etc. Pero no solo es que no encontremos ningún fichero útil, sino que comprobamos que todas esas carpetas están vacías y eso resulta un tanto sospechoso en un disco principal de un sistema operativo que no está recién instalado.

Por tanto, nunca está de más intentar recuperar en la medida de lo posible los ficheros borrados con una utilidad como [Restoration](#) (vale, no es que sea la mejor del mercado, pero es gratuita y no está mal para lo poco que necesitamos en este momento). Vemos que consigue recuperar bastantes ficheros, entre los que destacan bastantes que son relativamente recientes y cuya ubicación original era precisamente la carpeta de archivos temporales de Internet del usuario administrador.

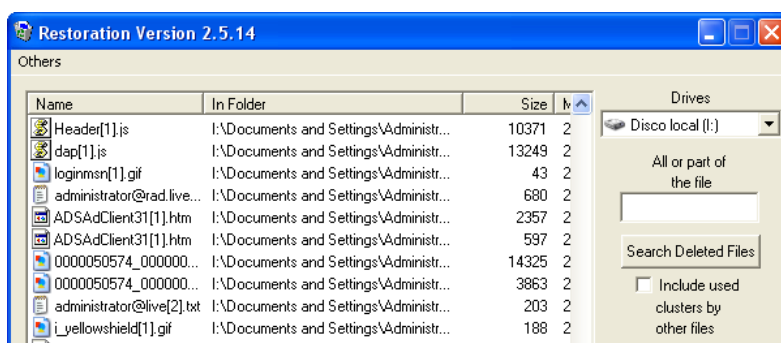


Fig. 5 – Restaurando archivos eliminados con Restoration

De momento localizamos al menos uno de ellos que parece contener alguna pista interesante. Se trata del fichero *InboxLight[1].htm*, que contiene una vista de la bandeja de entrada del correo electrónico del usuario, en la que podemos ver que hay un mensaje de correo enviado por Juan Garrido con el asunto "*Mensaje importante*". Sin embargo, en el resto de ficheros recuperados no encontramos el contenido de dicho mensaje, pero sí que hay un ejecutable eliminado que llama la atención. Se trata de la utilidad [NotMyFault.exe](#) de Sysinternals, utilizada para forzar volcados de memoria al provocar fallos intencionados y controlados a través de su driver.

Llegados a este punto, conviene recordar la existencia de los vídeos y de las presentaciones del evento [Asegur@IT II](#), para los que no pudimos asistir. El vídeo de la sesión de Juan Garrido está cortado, pero entre el vídeo parcial y la presentación completa tenemos más que suficiente como para saber por dónde debemos continuar.

Encontramos en el disco virtual varios volcados de memoria dentro del directorio de Windows, pero destaca el *MEMORY.DMP*, que es un volcado completo (en el directorio *Minidump* hay otros dos que no nos servirán). Opcionalmente, podemos pasarle la utilidad [strings](#) de Sysinternals, que dejaría únicamente las cadenas de texto imprimible (ASCII y UNICODE), aunque su uso tampoco era necesario.

Si buscamos el contenido del texto "*Mensaje importante*" (por ejemplo, con un editor hexadecimal o con la utilidad de Windows [findstr](#)), localizaremos fácilmente un fragmento con el mensaje de correo electrónico, en el que se puede leer siguiente texto:

Ten mucho cuidado con este mensaje cifrado. No se lo dejes ver a nadie. Una vez lo hayas memorizado destruye todo archivo que relacione a este mail y al mensaje cifrado. Borra archivos temporales, cookies, ficheros del navegador y todo lo que se te ocurra para que no nos delate el equipo. La operación está en sus manos.

Saludos Sr. X

<Mensaje>Hjd+fAoAMTM2P2tUKi0qPC+Y+n47NBE3IDQ+LRs7O344M3s3P343FD...</Mensaje>

Es evidente que la última parte del mensaje contiene un fragmento codificado en Base-64, pero no conseguimos leerlo decodificándolo, por lo que parece que además de estar codificado, también está cifrado. Si recordamos que la cadena que obtuvimos al terminar la fase 1 estaba etiquetada como "clave de cifrado", bastará con aplicar un algoritmo XOR con aquella cadena `[/^*****^/]` a los bytes resultantes de la decodificación del texto. El descifrado se aplica exactamente igual que el de cifrado y consiste en aplicar el siguiente algoritmo:

```
PARA i = 1 HASTA longitud_texto
    cifrado(i) = texto(i) XOR clave(i MOD longitud_clave)
```

Aplicando este sencillo algoritmo al resultado de la decodificación del Base-64, obtendremos el siguiente texto en claro:

*El "Macaco" estará esperandote en la puerta del "Pub ** *****" el día de la "Luna de *****" a la hora del "Te*****".*

Indistintamente, para nuestro objetivo también podíamos haber usado el fichero *Pagefile.sys* del directorio raíz, que curiosamente contenía un texto diferente en el cuerpo del correo electrónico, con otro mensaje en Base-64 también diferente:

Ten mucho cuidado con este mensaje cifrado. No se lo dejes ver a nadie. Una vez lo hayas memorizado destruye todo archivo que relacione a este mail. Y cuidado con los archivos temporales, no vaya a ser que nos hagan un forense!
Saludos Sr. X
<Mensaje>Ag1SUDgRChUMKENSFwYECAas5kEXAQUVGxUBIw4GF1UVB1
QDJkECBxACHR...</Mensaje>

El texto resultó estar cifrado con una clave diferente (muy parecida, pero más corta y más simple). El texto descifrado resultó ser prácticamente el mismo (además del mostrado antes, tenía un salto de línea y el texto "*Saludos Malignos!*" al final). No obstante, a pesar de no haber tenido la contraseña o clave, este tipo de cifrado XOR se podría romper con mucha facilidad al estar aplicado sobre un texto plano de longitud considerable, aunque creo que eso lo dejaremos para otra ocasión si surge la oportunidad en un nuevo reto... ;-)

Por cierto, como es evidente, el reto termina al introducir literalmente los cuatro valores solicitados, que son los que aparecen entre comillas en nuestro texto descifrado.

Agradecimientos y comentarios

Como siempre, quiero terminar agradeciendo el trabajo de todos los que han hecho posible este reto, esta vez con mención especial a Juan Garrido (*Silverhack*), no solo por haber participado en el desarrollo del mismo, sino también porque gracias al vídeo y a las diapositivas de su charla, la segunda fase se me hizo mucho más sencilla.

No he incluido las diferentes contraseñas y claves que han ido apareciendo en las diferentes fases del reto, al igual que tampoco están los mensajes completos, porque considero que la única forma de asimilar este tipo de conocimientos es ponerse manos a la obra y practicar un poco, así que aprovecha que el reto está en línea (al menos de vez en cuando) y ámate a intentarlo.

Saludos,

Daniel Kachakil

dani@kachakil.com