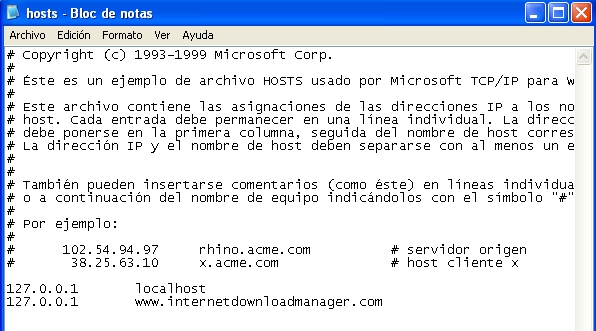
  
  
  
  
Ingeniería Inversa con Internet Download Manager  
PARTE II

Autor : Jmissing87  
Contacto : Jmissing87@gmail.com  
 www.twitter.com/Jmissing87

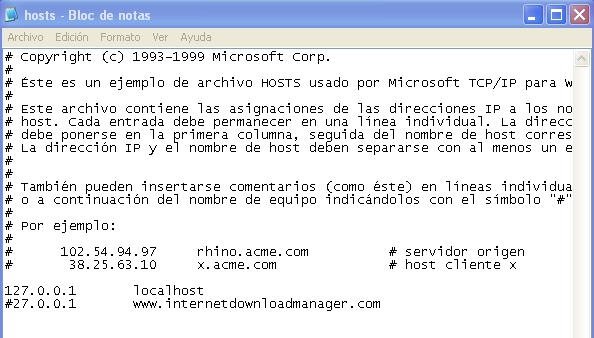
Software : Internet Download Manager  
Version : 6.23  
S.O : Windows XP sp3

Analizando el problema:

Inicio:  
En la parte anterior quedamos en que habíamos logrado generar un keygen con el cual podemos registrarnos sin ni un problema aparente, el problema comienza cuando ***IDM*** válida el serial registrado contra el servidor ***www.internetdownloadmanager.com***, la solución que se utilizaba antiguamente era modificar el archivo hosts ubicado en la ruta ***C:\WINDOWS\system32\drivers\etc\hosts*** añadiendo una línea al archivo.



Si ahora vamos a ***IDM*** y nos registramos.  


Nos aparecerá el siguiente mensaje.  
  
  
  
Si presionamos “***CONTINUAR***” .  
  
  
  
Ahora revisemos el archivo ***hosts*** para saber como lo ha dejado ***IDM*** .  
  
  
  
IDM ha comentado la línea que corresponde a su dominio, por lo tanto la resolución a su dominio la ha realizado a través del DNS y no del archivo HOSTS como queríamos, ahora cerramos el proceso de IDM, quitamos el signo # del archivo y lo cambiamos por un 1.  
  
Evitando CreateFile :  
  
Cargamos IDM con IMMUNITY y presionamos F9 , nos aparecerá nuevamente el error del serial y la ventana para registrarnos, antes de ingresar los datos en el formulario, presionamos CTRL+N para listar las apis utilizadas por IDM.  
  
Tipeamos CreateFile y nos aparecerán 2 resultados

***CreateFileA***

  
  
***CreateFileW***  
  
  
¿La diferencia?, ambas son similares la diferencia es que CreateFileA soporta caracteres ASCII y CreateFileW soporta caracteres UNICODE.  
  
La documentación de ésta función es la siguiente :  
  
**HANDLE** CreateFile(

**LPCTSTR** lpszName,

**DWORD** fdwAccess,

**DWORD** fdwShareMode,

**LPSECURITY\_ATTRIBUTES** lpsa,

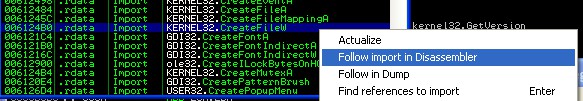
**DWORD** fdwCreate,

**DWORD** fdwAttrsAndFlags,

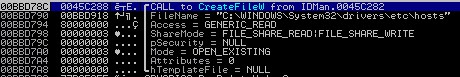
**HANDLE** hTemplateFile

);  
  
El Parámetro que nos interesa es el primero LPCTSTR lpszName donde se encuentra el nombre del archivo.

Pondremos un Breakpoint en ambas .



Luego ***F2***.  
  
Una vez que configuramos el BP en ambas funciones, comenzaremos a rellenar los datos del formulario y presionamos aceptar.

El programa parará justo cuando ***IDM*** se encuentre abriendo un archivo, en nuestro caso tenemos un poco de suerte ya que es el primer archivo que se lee es el ***HOSTS***.  
  
Sacamos los breakpoint y presionamos ***CTRL+F9***, guardamos el contenido del registro ***EAX*** que hace referencia al identificador del archivo en mi caso.  
  
**EAX** = 00000308

Salimos del ***CALL*** con***F8*** y vemos como continua esto.  
  
Vienen ***2*** llamadas a funciones que a nosotros no nos importan, a nosotros nos importa seguir la huella del ***IDENTIFICADOR***del archivo ***HOSTS***.

**0045C28E** |> FF15 D4286100 **CALL DWORD PTR DS:[<&ole32.CoUninitialize>]** ; ole32.CoUninitialize

**0045C294** |> 55 **PUSH EBP** ; /hLibModule

**0045C295** |. FF15 DC226100 **CALL DWORD PTR DS:[<&KERNEL32.FreeLibrary>]**  ; \FreeLibrary

Tenemos una copia del ***IDENTIFICADOR*** en ***EAX*** y ***EBX*.**  
  
**0045C28C** |. 8BD8 **MOV EBX,EAX**

El programa comprueba que al abrir el archivo no exista un error y lo comparará con -1 .  
  
**0045C29B** |. 83FB FF **CMP EBX,-1**

De lo contrario saltará para preparar el ***STACK*** y luego llamar al ***API******GetFileSize***.  
  
*Información de* ***GetFileSize*** :  
  
**DWORD** GetFileSize(

**HANDLE** hFile, // handle of file to get size of

**LPDWORD** lpFileSizeHigh, // address of high-order word for file size

);

En la llamada de ***IDM*** nos encontramos lo siguiente.  
  
**0045C2AA**|> 8D8424 8800000> **LEA EAX,DWORD PTR SS:[ESP+88]**

….

**0045C2B9**|. 50 **PUSH EAX ; /pFileSizeHigh**

**0045C2BA**|. 53 **PUSH EBX ; |hFile**….  
….

**0045C2C6**|. FF15 58246100 **CALL DWORD PTR DS:[<&KERNEL32.GetFileSize>]**

En ***EAX*** se almacena la dirección de una variable, la cual después de ejecutar la función nos entregará el resultado, en ***EBX*** tenemos el ***IDENTIFICADOR DEL ARCHIVO HOSTS***.  
  
Paremos después del ***API*** y analicemos el resultado y qué operación se realiza con los resultados.  
  
***EAX*** toma el valor ***00000345*** que en decimal corresponde a ***837***, el tamaño del archivo es de ***837bytes***, tenemos que en .

EAX => Tamaño del Archivo.   
EBP => Copia de EAX.  
EBX => Identificador del Archivo.  
  
Se comprueba nuevamente que el retorno de la función no sea un error, para ello lo hace comparando el retorno del api con -1 .  
  
**0045C2CC**|. 8BE8 **MOV EBP,EAX**

**0045C2CE**|. 83FD FF **CMP EBP,-1**

**0045C2D1**|. 75 0E **JNZ SHORT IDMan.0045C2E1**

Ahora saltamos a otra función llamada CreateFileMapping , ésta función recibirá como parámetro el IDENTIFICADOR del archivo HOSTS y el tamaño de nuestro archivo entre otros, la finalidad de ésta es cargar nuestro archivo en memoria, por lo tanto la función nos entregará otro identificador con el archivo en memoria, necesitamos poner atención a éste ahora, nos olvidamos del anterior, entonces después de ejecutar ese CALL guardamos el valor de EAX.  
  
**EAX** = 00000324  
  
Nuevamente abajo nos encontramos con una comparación y un salto para asegurar que no exista un error en el retorno de la función.  
  
**0045C2F5**|. 8BF0 **MOV ESI,EAX**

**0045C2F7**|. 3BF7 **CMP ESI,EDI**

**0045C2F9**|. 897424 48 **MOV DWORD PTR SS:[ESP+48],ESI**

**0045C2FD**|. 0F84 CC010000 **JE IDMan.0045C4CF**

Si el programa no salta continuará con la siguiente ***API*** esta api recibe como parámetro el ***IDENTIFICADOR*** anterior entregado y el tamaño, ésta función nos retorna un puntero a un buffer con el contenido del archivo, por supuesto que debemos seguir la pista de ahora en adelante **;)** .  
hy   
**0045C303**|. 57 **PUSH EDI ; /MapSize**

**0045C304**|. 57 **PUSH EDI ; |OffsetLow**

**0045C305**|. 57 **PUSH EDI ; |OffsetHigh**

**0045C306**|. 6A 04  **PUSH 4 ; |AccessMode = FILE\_MAP\_READ**

**0045C308**|. 56 **PUSH ESI ; |hMapObject**

**0045C309**|. FF15 60246100 **CALL DWORD PTR DS:[<&KERNEL32.MapViewOfFile>]; \MapViewOfFile**

El contenido del archivo ***HOSTS*** ahora se guarda en la dirección .  
  
**EAX** = 010D0000  
  
Continuamos y vemos que se copia el puntero del buffer al registro ***EDI*** y luego se testea que éste valor no sea ***0x0***, de lo contrario llamará a una función propia y que recibe como único parámetro ***EBP+1*** recordemos que en ***EBP*** se almacena una copia del tamaño del archivo en éste caso al tamaño le suma **1** y lo pasa como argumento.  
  
**0045C31D**|. 8D75 01 **LEA ESI,DWORD PTR SS:[EBP+1]**

**0045C320**|. 56 **PUSH ESI**

**0045C321|**. E8 96391800 **CALL IDMan.005DFCBC**

**0045C326|**. 56 **PUSH ESI**

**0045C327|**. 8BD8 **MOV EBX,EAX**

**0045C329|**. E8 8E391800 **CALL IDMan.005DFCBC**

Los dos ***CALL*** que vemos hacen referencia a la misma función y la función de éstas son crear un espacio(Buffer) en el ***HEAP*** del tamaño del archivo ***HOSTS***(0x345bytes aprox.) que tenemos en nuestro ***S.O***, por lo tanto ejecutaremos los ***CALL’s*** anotando los valores de retorno que se encuentran en ***EAX***.  
  
**Buffer\_1** : 0158CEA0  
**Buffer\_2** : 0158D208  
  
Una vez tenemos las direcciones, sigamos revisando las instrucciones.  
  
**0045C32E** |. 83C4 08 **ADD ESP,8**

**0045C331** |. 894424 10  **MOV DWORD PTR SS:[ESP+10],EAX**

**0045C335** |. 85DB **TEST EBX,EBX**

**0045C337** |. 0F84 7A010000 **JE IDMan.0045C4B7**

**0045C33D** |. 85C0 **TEST EAX,EAX**

**0045C33F** |. 0F84 72010000 **JE IDMan.0045C4B7**

Podemos ver que la dirección del **buffer\_2** contenido en ***EAX*** se almacena en la variable ***[ESP+10]*** que corresponde a la dirección.  
  
**$+10** > 0158D208 ÒX

Luego se realiza el testeo para saber si se ha reservado memoria y continuar con el programa, en ésta ocasión tenemos en ***EBX*** la dirección del primer buffer y en ***EAX*** la del segundo buffer.  
  
Una vez que reservamos espacio, se prepara la pila para llamar otra función que desconocemos lo que haces, analicemos ***SU PARÁMETRO***.

**0045C345** |. 8D5424 50  **LEA EDX,DWORD PTR SS:[ESP+50]**  
**0045C34E** |. 52 **PUSH EDX**

Toma la dirección de memoria ***00BBD7FC*** desde la pila para luego meter esa dirección de memoria en la pila ***COMO PARÁMETRO DE LA FUNCIÓN QUE LLAMARÁ***, ahora en esa dirección de memoria se comienza a meter muchos caracteres que más adelante veremos que significa ;) .

**0045C349** |. C64424 50 21 **MOV BYTE PTR SS:[ESP+50],21**

**0045C34F** |. C64424 55 66 **MOV BYTE PTR SS:[ESP+55],66**

**0045C354** |. C64424 56 61 **MOV BYTE PTR SS:[ESP+56],61**

**0045C359** |. C64424 57 7B **MOV BYTE PTR SS:[ESP+57],7B**

**0045C35E** |. C64424 58 6A **MOV BYTE PTR SS:[ESP+58],6A**

**0045C363** |. C64424 59 7D  **MOV BYTE PTR SS:[ESP+59],7D**

**0045C368** |. C64424 5A 61 **MOV BYTE PTR SS:[ESP+5A],61**

**0045C36D** |. C64424 5B 6A **MOV BYTE PTR SS:[ESP+5B],6A**

**0045C372** |. C64424 5C 7B **MOV BYTE PTR SS:[ESP+5C],7B**

**0045C377** |. C64424 5D 6B **MOV BYTE PTR SS:[ESP+5D],6B**

**0045C37C** |. C64424 5E 60 **MOV BYTE PTR SS:[ESP+5E],60**

**0045C381** |. C64424 5F 78  **MOV BYTE PTR SS:[ESP+5F],78**

**0045C386** |. C64424 60 61 **MOV BYTE PTR SS:[ESP+60],61**

**0045C38B** |. C64424 61 63  **MOV BYTE PTR SS:[ESP+61],63**

**0045C390** |. C64424 62 60  **MOV BYTE PTR SS:[ESP+62],60**

**0045C395** |. C64424 63 6E  **MOV BYTE PTR SS:[ESP+63],6E**

**0045C39A** |. C64424 64 6B  **MOV BYTE PTR SS:[ESP+64],6B**

**0045C39F** |. C64424 65 62  **MOV BYTE PTR SS:[ESP+65],62**

**0045C3A4** |. C64424 66 6E  **MOV BYTE PTR SS:[ESP+66],6E**

**0045C3A9** |. C64424 67 61 **MOV BYTE PTR SS:[ESP+67],61**

**0045C3AE** |. C64424 68 6E  **MOV BYTE PTR SS:[ESP+68],6E**

**0045C3B3** |. C64424 69 68  **MOV BYTE PTR SS:[ESP+69],68**

**0045C3B8** |. C64424 6A 6A **MOV BYTE PTR SS:[ESP+6A],6A**

**0045C3BD** |. C64424 6B 7D  **MOV BYTE PTR SS:[ESP+6B],7D**

**0045C3C2** |. C64424 6C 21  **MOV BYTE PTR SS:[ESP+6C],21**

**0045C3C7** |. C64424 6D 6C **MOV BYTE PTR SS:[ESP+6D],6C**

**0045C3CC** |. C64424 6E 60  **MOV BYTE PTR SS:[ESP+6E],60**

**0045C3D1** |. C64424 6F 62  **MOV BYTE PTR SS:[ESP+6F],62**

**0045C3D6** |. C64424 70 00 **MOV BYTE PTR SS:[ESP+70],0**

Pongamos un ***BP*** en el ***CALL*** y luego ***F9***, luego nos metemos a éste con ***F7***, en ésta función lo primero que vemos es un ***MOV*** desde ***[esp+0x4]*** es decir el parámetro que introdujimos, si vamos a esa dirección en la pila veremos la ristra de caracteres que se ingresaron anteriormente.  
  
**[ESP+0x4]** = 00BBD7FC  
  
**00BBD7FC** 21 66 61 7B 6A 7D 61 6A  **!fa{j}aj**

**00BBD804** 7B 6B 60 78 61 63 60 6E **{k`xac`n**

**00BBD80C** 6B 62 6E 61 6E 68 6A 7D  **kbnanhj}**

**00BBD814** 21 6C 60 62  **!l`b**

Vemos que pasa después, de ***PUSH*** a ***POP*** podemos ver un loop

**0059F684** |. 57 **PUSH EDI**

**0059F685** |. 8BFA **MOV EDI,EDX**

**0059F687** |. 83C9 FF **OR ECX,FFFFFFFF**

**0059F68A** |. 33C0 **XOR EAX,EAX**

**0059F68C** |. F2:AE  **REPNE SCAS BYTE PTR ES:[EDI]**

**0059F68E** |. F7D1 **NOT ECX**

**0059F690** |. 49  **DEC ECX**

**0059F691** |. 5F  **POP EDI**

Ese loop no hace más que contar la cantidad de caracteres que contiene la cadena que vimos anteriormente, luego pasamos a otro loop, el siguiente loop utiliza el registro ***EAX*** como índice para ir recorriendo la cadena que se encuentra en ***EDX*** , el loop terminará cuando termine de recorrer la cadena completa, pero mientras recorre la cadena completa se realiza un ***XOR*** con el valor ***0xF*** a cada uno de los caracteres, si vamos a la dirección de ***EDX*** con el boton derecho ***“FOLLOW IN DUMP”*** y luego ponemos un breakpoint en.  
  
**0059F6A5** |. 5B  **POP EBX**

Presionamos ***F9*** y Podemos ver el resultado del ***XOR 0xF*** .  
  
**00BBD7FC** 2E 69 6E 74 65 72 6E 65 **.interne**

**00BBD804** 74 64 6F 77 6E 6C 6F 61 **tdownloa**

**00BBD80C** 64 6D 61 6E 61 67 65 72 **dmanager**

**00BBD814** 2E 63 6F 6D **.com**

Acabamos de descubrir una string oculta que posiblemente sea la que se utilice para realizar una comparación en el buffer donde se almacena el archivo ***HOSTS***, vamos a parchar el ejecutable para corroborar que es ahí donde se comprueba el dominio, modificando una letra del string podremos pasar la revisión del archivo **HOSTS**.  
  
Vamos a la dirección donde se pasaba la cadena antes de ***SER DESCIFRADA*** con el ***XOR***, tomamos cualquiera de sus ***MOV*** y anotamos los ***OPCODES***.

**0045C349** |. C64424 50 21 **MOV BYTE PTR SS:[ESP+50],21**

**0045C34F** |. C64424 55 66 **MOV BYTE PTR SS:[ESP+55],66**

**0045C354** |. C64424 56 61 **MOV BYTE PTR SS:[ESP+56],61**

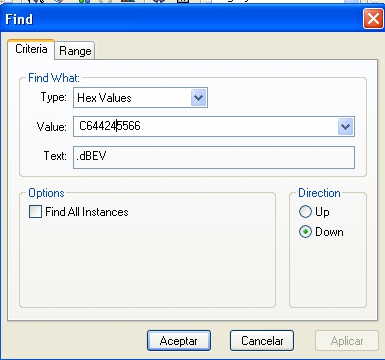
**0045C359** |. C64424 57 7B **MOV BYTE PTR SS:[ESP+57],7B**

En mi caso tomaré los ***OPCODES*** de la dirección **0045C34F** y que corresponde a **C64424 55 66**, sabemos que ese **66** corresponde a un valor con ***XOR 0xf*** veamos a que caracter corresponde realizando la función ***XOR 0x66,0xf***.

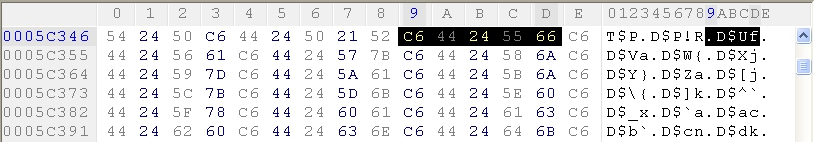


Sabemos que el valor ***0x69*** corresponde a la letra **‘i’** , cambiaremos esa letra por una **‘b’** su valor hexadecimal es ***0x62*** por lo tanto aplicaremos un ***XOR 0x63,0xf*** y ésto nos da como resultado ***0x6c***.  
  
Debemos modificar :  
  
 **C64424 55 66**

por  
 **C64424 55 6c**

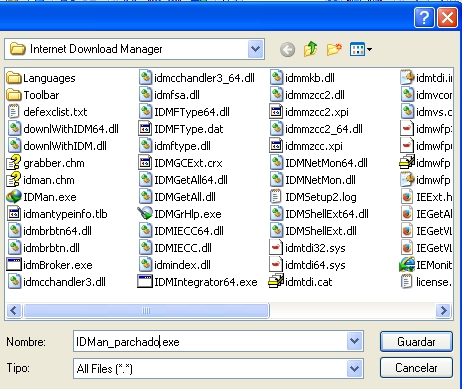
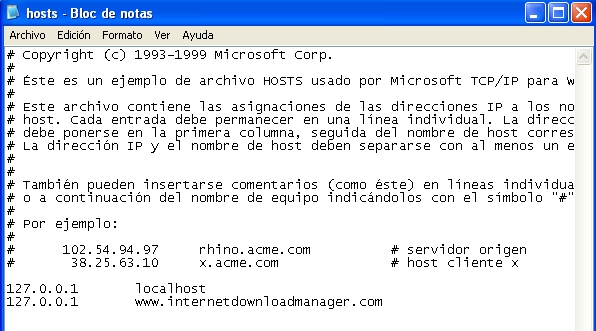
Cerramos immunity y abrimos el ejecutable de IDM con **“Hex WorkShop”** con éste editor hexadecimal buscaremos los opcodes anteriores y reemplazaremos el byte.  
  


Y encontramos los **OPCODES**.

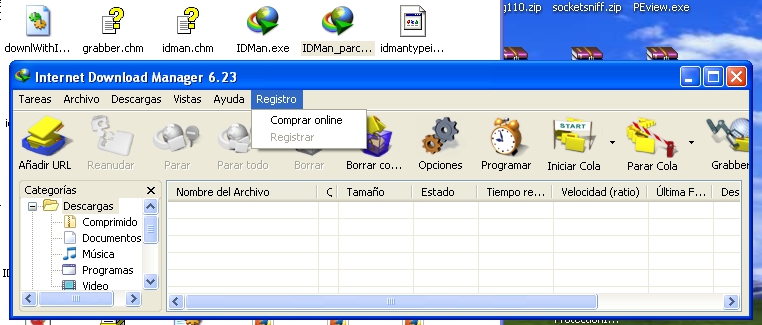


Modificamos el 0x66 por 0x6c .



y guardamos el IDM parchado.  
  
  
  
  
Antes de ejecutar ese ***IDMan\_parchado.exe*** revisemos el archivo hosts que se encuentre la línea de ***internetdownloadmanager***.  
  
  


Ahora ejecutemos.

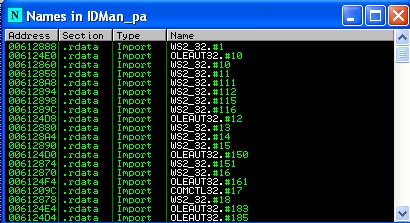


PERO seguimos sin registrar 100% nuestro IDM ésto porque si reiniciamos la máquina y nuestro idm\_parchado veremos que nos aparece el siguiente mensaje.

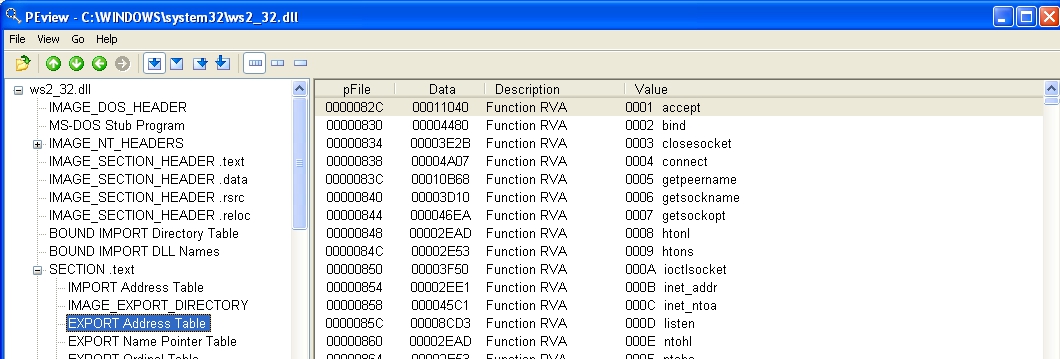


Luego se nos cierra nuevamente IDM, el esfuerzo anterior que hicimos al realizar la modificación del string y evitar la modificación del HOSTS no ha sido para nada infructuoso, lo que necesitamos ahora es saber cómo se comunica IDM con el servidor, que cadena espera desde el servidor y cómo se valida.

Analizando SOCKET en IDM:

Para analizar las conexiones que realiza IDM necesitamos descomentar la línea del archivo HOSTS que corresponde a IDM, luego cargamos nuestro IDM parchado, buscamos las api relacionadas con **ws2\_32.dll** que es la librería que se utiliza para manejar los socket en windows.  
  


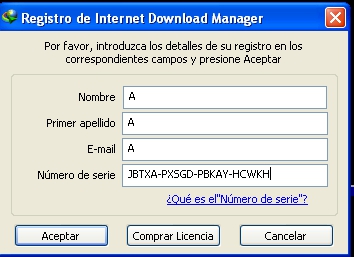
Nos encontramos con la sorpresa de que esas funciones han sido importadas por su ORDINAL y no por el nombre, para conseguir el nombre vamos a utilizar el programa **PeView.exe** que nos ayudará a descubrir el ordinal correspondiente al nombre de la función, cargamos la **ws2\_32.dll** en **PeView.exe**.



¿Qué funciones necesitamos interceptar?  
  
 **Nombre:** **HEXADECIMAL:** **DECIMAL:**  
1.- connect() 0x04 4  
2.- send() 0x13 19  
3.- recv() 0x10 16  
  
La información de las funciones:  
  
*int* ***connect****(  
 \_In\_* ***SOCKET*** *s,  
 \_In\_* ***const struct sockaddr*** *\*name,  
 \_In\_* ***int*** *namelen  
);*

*int* ***send****(  
 \_In\_*  ***SOCKET*** *s,  
 \_In\_ const* ***char*** *\*buf,  
 \_In\_* ***int***  *len,  
 \_In\_* ***int*** *flags  
);*

*int* ***recv****(  
 \_In\_* ***SOCKET*** *s,  
 \_Out\_* ***char*** *\*buf,  
 \_In\_* ***int*** *len,  
 \_In\_* ***int*** *flags  
);*

Pongamos un BP en ***ws2\_32.#4*** , ***ws2\_32.#19*** y ***ws2\_32.#16*** , en **PeView** obtenemos el número en hexadecimal, debemos pasarlo a decimal.  
  
Una vez configurado los breakpoint’s corremos nuestro programa, éste parará para pedirnos los datos de registro, ingresamos cualquier dato pero un serial valido.  
  
  
  
Ahora al presionar “aceptar” el programa parará en connect(), es importante analizar los datos de ésta función ya que a través de él parámetro Nº2 podremos saber la dirección IP y el PUERTO al que se conecta para realizar la validación.  
  
  
  
En el segundo parámetro presionamos “Follow in Dump” con ésto llegamos a la estructura sockaddr que contiene los siguientes miembros.

struct sockaddr {

unsigned short sa\_family; // address family, AF\_xxx

char sa\_data[14]; // 14 bytes of protocol address

};

Tomamos los primeros 18 bytes.  
  
**013FE480** 02 00 00 50 A9 37 00 E0 ..P©7.à

**013FE488** D8 FF 3F 00 00 00 00 00 Øÿ?.....

Los dos primeros corresponden al miembro sa\_family y los 2 siguientes equivalen al puerto.  
  
**sa\_family** = 02 00  
**puerto** = 00 50 => El 0x50 corresponde a 80 en decimal ;) .  
  
Los próximos bytes corresponden a la dirección IP en hexadecimal.  
  
**A9** = 169  
**37** = 55  
**00** = 0  
**E0** = 224  
  
Tenemos que el **IDM** se conecta a **169.55.0.224** a través de el puerto **80**.  
  
Ya sabemos a qué servidor conecta y a que puerto, ahora presionamos F9 y paramos justo antes de enviar la primera petición HTTP, es decir en la función send(), es importante tomar nota de la cantidad de bytes que envía y el contenido del buffer enviado.  
  
  
  
**Dir. Buffer** = 0x0156B680  
**Tamaño** = 0x18C es decir 396bytes  
**Contenido buffer** =  
  
**0156B680** 50 4F 53 54 20 2F 64 61 POST /da

**0156B688** 74 61 2F 33 39 35 30 31 ta/39501

**0156B690** 32 37 31 32 2F 72 65 67 2712/reg

**0156B698** 69 73 74 65 72 2E 63 67 ister.cg

**0156B6A0** 69 20 48 54 54 50 2F 31 i HTTP/1

**0156B6A8** 2E 31 0D 0A 55 73 65 72 .1..User

**0156B6B0** 2D 41 67 65 6E 74 3A 20 -Agent:

**0156B6B8** 4D 6F 7A 69 6C 6C 61 2F Mozilla/

**0156B6C0** 34 2E 30 20 28 63 6F 6D 4.0 (com

**0156B6C8** 70 61 74 69 62 6C 65 3B patible;

**0156B6D0** 20 4D 53 49 45 20 38 2E MSIE 8.

**0156B6D8** 30 3B 20 57 69 6E 64 6F 0; Windo

**0156B6E0** 77 73 20 4E 54 20 35 2E ws NT 5.

**0156B6E8** 31 3B 20 54 72 69 64 65 1; Tride

**0156B6F0** 6E 74 2F 34 2E 30 29 0D nt/4.0).

**0156B6F8** 0A 48 6F 73 74 3A 20 77 .Host: w

**0156B700** 77 77 2E 69 6E 74 65 72 ww.inter

**0156B708** 6E 65 74 64 6F 77 6E 6C netdownl

**0156B710** 6F 61 64 6D 61 6E 61 67 oadmanag

**0156B718** 65 72 2E 63 6F 6D 0D 0A er.com..

**0156B720** 41 63 63 65 70 74 3A 20 Accept:

**0156B728** 2A 2F 2A 0D 0A 41 63 63 \*/\*..Acc

**0156B730** 65 70 74 2D 4C 61 6E 67 ept-Lang

**0156B738** 75 61 67 65 3A 20 65 73 uage: es

**0156B740** 2D 45 53 0D 0A 41 63 63 -ES..Acc

**0156B748** 65 70 74 2D 43 68 61 72 ept-Char

**0156B750** 73 65 74 3A 20 2A 0D 0A set: \*..

**0156B758** 52 65 66 65 72 65 72 3A Referer:

**0156B760** 20 68 74 74 70 3A 2F 2F http://

**0156B768** 77 77 77 2E 69 6E 74 65 www.inte

**0156B770** 72 6E 65 74 64 6F 77 6E rnetdown

**0156B778** 6C 6F 61 64 6D 61 6E 61 loadmana

**0156B780** 67 65 72 2E 63 6F 6D 2F ger.com/

**0156B788** 64 61 74 61 2F 33 39 35 data/395

**0156B790** 30 31 32 37 31 32 2F 0D 012712/.

**0156B798** 0A 41 75 74 68 6F 72 69 .Authori

**0156B7A0** 7A 61 74 69 6F 6E 3A 20 zation:

**0156B7A8** 42 61 73 69 63 20 64 47 Basic dG

**0156B7B0** 4E 70 5A 47 30 36 59 58 NpZG06YX

**0156B7B8** 70 6A 5A 6D 64 77 5A 57 pjZmdwZW

**0156B7C0** 51 3D 0D 0A 43 6F 6E 74 Q=..Cont

**0156B7C8** 65 6E 74 2D 54 79 70 65 ent-Type

**0156B7D0** 3A 20 61 70 70 6C 69 63 : applic

**0156B7D8** 61 74 69 6F 6E 2F 78 2D ation/x-

**0156B7E0** 77 77 77 2D 66 6F 72 6D www-form

**0156B7E8** 2D 75 72 6C 65 6E 63 6F -urlenco

**0156B7F0** 64 65 64 0D 0A 43 6F 6E ded..Con

**0156B7F8** 74 65 6E 74 2D 4C 65 6E tent-Len

**0156B800** 67 74 68 3A 20 34 39 36 gth: 496

**0156B808** 0D 0A 0D 0A 00 F0 AD BA .....ð­º

…….  
  
El buffer es más largo pero solo nos importan esos bytes.  
  
  
Nuevamente presionamos F9 y parará otra vez en la función send(), ésta vez envía los datos que deberían ir adjuntos a la petición HTTP de tipo POST es decir las variables.  
  
  
  
**Dir. Buffer** = 0x0158D240  
**Tamaño** = 0x1F0 es decir 496bytes  
**Contenido buffer** =  
  
**0158D240** 22 7C 0A 59 A2 7B CC 94 "|.Y¢{Ì”

**0158D248** 42 AC 30 29 7E 30 1C C1 B¬0)~0Á

**0158D250** 08 E4 59 52 69 15 C1 CF äYRiÁÏ

**0158D258** 11 46 F9 48 4B B1 C8 04 FùHK±È

**0158D260** 6B AA 79 46 09 AF 87 7E kªyF.¯‡~

**0158D268** 79 70 C7 54 FB 54 1E 98 ypÇTûT˜

**0158D270** 3A 6E 82 F7 7F 94 E3 F8 :n‚÷”ãø

**0158D278** 24 8E DE 2E DF 17 B4 BC $ŽÞ.ß´¼

**0158D280** 65 9D 6C 3A 81 D3 A5 C4 el:Ó¥Ä

**0158D288** C6 1F F5 9A DD 23 4B CE ÆõšÝ#KÎ

**0158D290** C0 68 3D 2B AE 11 29 93 Àh=+®)“

**0158D298** 02 DC 76 9E F2 C0 04 DD ÜvžòÀÝ

**0158D2A0** B9 99 30 BF 12 F7 75 1E ¹™0¿÷u

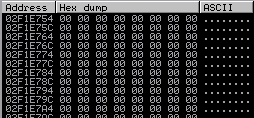
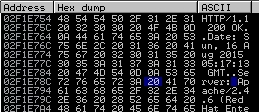
**0158D2A8** C8 31 DA 13 80 D0 83 0A È1Ú€Ðƒ.

**0158D2B0** 4C 92 06 46 FD BD 1F 9F L’Fý½Ÿ

**0158D2B8** C1 76 C0 9A F6 D2 56 4A ÁvÀšöÒVJ

**0158D2C0** 92 7E 7F 9E 96 D5 2B 6A ’~ž–Õ+j

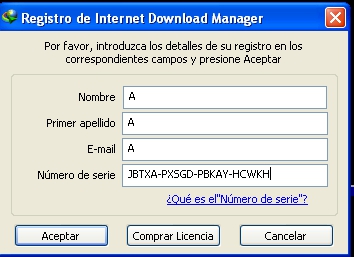
**0158D2C8** 46 8F 3C F2 5C 68 EE 21 F<ò\hî!

También debemos anotar el tamaño y el contenido del buffer ya que posteriormente analizaremos estos datos.  
  
Presionamos F9 y parará en la función recv(), realizamos el mismo procedimiento de antes con el buffer, con la diferencia que ahora en el buffer no nos aparecerá nada pero si presionamos CTRL+F9 para que se ejecute la función, se nos llenará el buffer.  
  
  
  
buffer sin info.  
  
  
Presionamos CTRL+F9  
  
  
**Dir. Buffer** = 0x02F1E754  
**Tamaño** = 0x200 es decir 512bytes  
**Contenido buffer** =  
  
Presionamos F9 y vemos que el envío y recibo de datos ha terminado, nos aparecerá el siguiente mensaje.  
  
  
  
  
Lo importante de todo ésto que acabamos de ver es analizar los datos de los buffer que obtuvimos ya que a través de ésto podremos seguir “Atacando” el software y continuar el análisis.  
  
Análisis:  
---------  
  
1.- Lo primero que salta a la vista es la dirección de los buffer’s, a través de éstas me puedo dar cuenta que el buffer para almacenar los datos de envío y recibo de datos se realizó en tiempo de EJECUCIÓN, es decir esos BUFFER’s se encuentran en una zona de memoria llamada HEAP, y como el buffer es generado en tiempo de ejecución la dirección de éstos debería cambiar cada vez que ejecutemos nuestro IDM(dificulta un tanto el proceso).  
  
Si buscamos en la lista de apis que utiliza IDM podremos encontrar la función.  
  
LPVOID HeapAlloc(

HANDLE hHeap, // handle to the private heap block

DWORD dwFlags, // heap allocation control flags

DWORD dwBytes // number of bytes to allocate   
);

Los buffer son creados con esa api, debemos recordarla **}:)** .  
  
  
2.- Lo segundo que debemos saber son los tamaños de los buffer que nos entregaron las funciones send() y recv().  
  
1er send() = 0x18C  
2do send() = 0x1f0  
 recv() = 0x200  
  
Ésta información la utilizaremos para tracear los buffer y saber los parámetros que se envían al servidor ésto porque a través de los tamaños podríamos poner BP condicionales que se activen cuando HeapAlloc() se llame y utilice como 3er parámetro cualquiera de esos valores :D .  
  
3.- Se utilizó dos veces la función send() la primera para enviar una petición HTTP que solo contiene los **METADATOS** para posteriormente con el otro send enviar los datos importantes.  
  
**Datos importantes Buffer 1**  
  
**a.-**Tipo de petición = **POST**  
Indica que los datos de las variables se enviará después del **HEADER HTTP**.  
  
**b.-** Ruta servidor = **/data/395012712/register.cgi**A través de la ruta sabemos el nombre del script que procesa las variables enviadas, es decir “register.cgi”.  
  
**c.-** Host = **www.internetdownloadmanager.com**El campo Host también me parece importante, esto porque una dirección ip puede albergar múltiples webs, a través de éste parámetro sabemos contra que web se realiza la petición.  
  
**d.-** Authorization = **Basic dGNpZG06YXpjZmdwZWQ=**Este campo nos indica que para acceder al script “register.cgi” debemos autenticarnos, ese tipo de autenticación será de tipo “Basic” esto significa que los datos de autenticación se enviarán encodeados con el algoritmo base64 y el formato “user:pass”, si pasamos el parámetro **dGNpZG06YXpjZmdwZWQ=** por el algoritmo base64 obtenemos.  
  
base64(**dGNpZG06YXpjZmdwZWQ=**) **---> tcidm:azcfgped**  
Es decir que los datos de autenticación son.  
  
user => tcidm  
pass => azcfgped  
  
**e.-** Content-Length = **Content-Length: 496**Este parámetro nos indica que terminado la petición HTTP se enviarán 496 bytes que contiene los datos obtenidos por IDM para realizar la validación a través de sus servidores. Por supuesto que estos datos se enviaron en el segundo send() .  
  
  
**Datos importantes Buffer 2**Acabamos de analizar el buffer\_1 que se envió a través del primer llamado a send(), dijimos que la cantidad de datos que se enviarían en el segundo llamado eran 496bytes, pues éste es un número mentiroso ésto porque si revisamos los datos del buffer efectivamente nos toparemos con esa cantidad de bytes enviados pero si analizamos vemos que los primeros 136bytes son los que tienen datos importantes y el resto no contiene nada, ¿Por qué?, porque ese buffer depende de la cantidad de caracteres que ingresemos en el formulario de validación, yo ingrese los siguientes.  
  
  
  
Es decir 5\*4=20 que son los datos del serial más los 3 de los caracteres especiales(-) y debemos sumar los datos personales 3bytes(A), por lo tanto tenemos 26bytes seteados, PEROOOOO ALGO NO CUADRA, al parecer IDM está enviado más información aparte de la que nos solicita **:S**, lo peor de todo es que si miramos el **buffer\_2**.  
  
(No lo copie entero)  
**0158D240** 22 7C 0A 59 A2 7B CC 94 "|.Y¢{Ì”

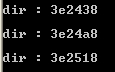
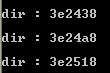
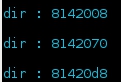
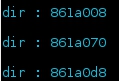
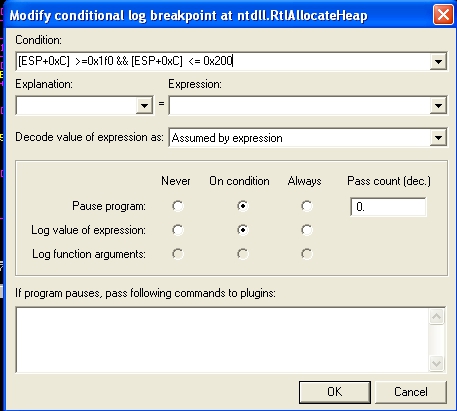
**0158D248** 42 AC 30 29 7E 30 1C C1 B¬0)~0Á

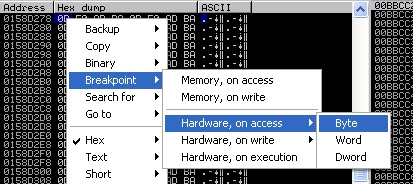
**0158D250** 08 E4 59 52 69 15 C1 CF äYRiÁÏ

**0158D258** 11 46 F9 48 4B B1 C8 04 FùHK±È

**0158D260** 6B AA 79 46 09 AF 87 7E kªyF.¯‡~

…….  
  
No tenemos datos legibles, al parecer los datos se encuentran cifrados.  
  
Para obtener los datos en Texto Plano, No podemos poner un BP en la dirección del buffer puesto que cuando recién nos enteramos de la dirección del buffer es cuando lo envía a través de send() y ya es demasiado tarde porque IDM ya los ha cifrado, tampoco podemos echar andar el IDM anotar la dirección del buffer y al abrir nuevamente IDM poner un BP antes que se cifre, ésto porque el buffer dijimos que es dinámico se crea en tiempo de ejecución, por ende la dirección cambia.  
  
Cuando digo que el buffer es dinámico lo es y su dirección también, pero a pesar de ser dinámico ¿podría ser predecible?, para realizar una prueba y verificar la teoría, compilamos y ejecutamos el siguiente código.  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main()  
{  
 int i=0;  
   
 while(i <= 2)  
 {  
 printf(“\ndir : %x\n”, malloc(sizeof(char)\*100));  
 i+=1;  
 }  
 return 0;

}  
  
Este código crea un buffer dinámico con la función malloc() de tamaño de 100 char, luego nos imprime por pantalla la dirección de memoria en el HEAP donde se ha creado el buffer, hagamos la siguiente prueba, ejecutemos una vez nuestro programa y tomamos nota de las direcciones que no aparecen(deben ser 3 direcciones), luego ejecutemos nuevamente y aparecerán 3 direcciones más, ahora comparamos los resultados, ¿se han obtenido los mismos valores en la primera y segunda ejecución?.  
  
Mis datos:  
  
En mi caso he realizado la prueba en un “Windows XP sp3” virtualizado con VBox y como sistema Base utilizo debian.  
  
Primera ejecución.  
  
  
Segunda ejecución.  
  
  
Como podemos ver en las dos ejecuciones nuestras direcciones son idénticas, por ende los buffer’s creados en el HEAP podrían ser PREDECIBLES ésto no ocurre siempre, pondré los resultados de mi S.O base que es debian.  
  
Primera ejecución.  
  
  
  
  
Segunda ejecución.  
  
  
Las direcciones son distintas :p .  
  
Solución:  
  
1.- Cargamos IDM en Immunity.  
2.- Presionamos F9.  
3.- Rellenamos los datos de validación sin presionar “aceptar”.  
4.- Presionamos CTRL+N y buscamos el api HEapAlloc().  
5.- Vamos a poner un “BP Conditional Log”  
6.- Ponemos como condición   
 [ESP+0xC] >=0x1f0 && [ESP+0xC] <= 0x200  
  
Con esto tomamos como condición que el 3er parámetro del api se encuentre entre 0x1f0 y 0x200, ¿Por qué?, porque si miramos el tamaño del buffer\_2 cuando se envía a través de send() es 0x1f0 es decir que como mínimo el tamaño 0x1f1, el buffer puede tener cualquier tamaño pero send() solo enviará esos bytes entonces pongo como límite 0x200(al OJO).  
  
También debemos indicar que el programa debe parar cuando se cumpla la condición.  
  
  
  
7.- Presionamos el botón aceptar y parará justo cuando se esté creando nuestro buffer para ser enviado :D .  
  
  
  
Presionamos CTRL+F9.

8.- Tomamos el valor de EAX que en mi caso es **0158D278** y ponemos “follow in dump”.  
  
  
  
9.- Tomamos 1 byte y ponemos un “Hardware On Write”.  
  
  
  
10.- Quitamos o deshabilitamos el “breakpoint conditional log” que pusimos anteriormente y presionamos **F9**.  
  
Hemos caído justo en la instrucción que se encuentra escribiendo datos en el buffer que hemos interceptado, pero como vemos es demasiado tarde, ya que los datos se están copiando cifrados.  
  
Immunity para en la instrucción.  
  
**005A6F67** . F3:A5  **REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]  
  
Estado de los registros:**

Ésta instrucción moverá tantos bloques de memoria de 4bytes desde [ESI] a [EDI] como lo indique el registro ECX, éste en mi caso dice “0x7B” para saber la cantidad exacta de datos que debe mover debemos realizar la siguiente operación (0x7b\*0x4)+0x4, ésto porque ECX desciende de 1 en 1 y la cantidad de bytes que se copian en cada uno son 4 a eso debemos sumar 0x4 puesto que pusimos un “HARDWARE BREAKPOINT” y éste para cuando se ha ejecutado la instrucción, si revisamos los datos en la direc de EDI-0x4 podemos ver que en nuestro buffer se han copiado 4 bytes.   
  
**0158D274**  22 7C 0A 59  **"|.Y**

0158D27C 0D F0 AD BA 0D F0 AD BA **.ð­º.ð­º**

Si miramos el contenido de la dirección de ESI podemos ver el buffer que se está copiando, ya sabemos el tamaño de éste realizando la operación anterior es 496bytes, por lo tanto tomamos el puntero ESI y para saber la dirección de comienzo le restamos 4bytes y si queremos saber la dirección de termino le sumamos 496bytes a la dirección de inicio.  
  
**INICIO :** 0158CEB0 - 4 = 0158CEAC  
**TERMINO :** 0158CEAC + 496 = 0158D09C  
  
Si miramos los datos entre esas direcciones podemos ver que los datos cifrados que nos importan son los primeros bytes, el tamaño de éstos es variable y lo veremos ahora, guardamos nuestra dirección **ESI**=0158CEAC.  
  
Cerramos y ejecutamos nuevamente IDM, presionamos F9 ingresamos los datos de registro pero antes de presionar “aceptar” en el DUMP buscamos la dirección de nuestro anterior **ESI**=0158CEAC, ponemos un “HARDWARE BREAKPOINT ON WRITE” en un BYTE presionamos F9 YYYYY.

  
  
Tenemos Nuestro Buffer completo descifrado **:D**.  
  
IMPORTANTE:  
  
Es importante que para realizar el proceso correctamente, el test que hicimos con la función malloc() se ha realizado correctamente, recuerden chequear las direcciones que nos entrega, por otro lado también es importante definir un SERIAL y los datos de pruebas, no cambiar éstos mientras estamos realizando el análisis por ejemplo mis datos son los siguientes.  
  
**Nombre** : A  
**Apellido** : A  
**Email** : A  
**Serial** : AAAAA-AAAAA-AAAAA-AAAAA  
  
El serial que he utilizado pasa tranquilamente por el validador, ésto porque he parchado IDM para que lo haga(Lo hice para realizar pruebas), pero si has definido un serial valido generado con nuestro keygen, no tendremos problemas.  
  
  
Ya tenemos nuestra cadena en texto plano **:D**, ya sabemos cual es la información que se envía al servidor, entre otras el nombre del equipo,grupo de trabajo, información del disco duro, versión de IDM etc … A mi parecer es demasiada información enviada.  
  
**Analizando información de recv()**:  
  
Éste ha sido el motivo del por qué me he demorado tanto en la segunda parte, no logré descifrar el tráfico que viene desde el servidor, son sólo 64 bytes pero como soy principiante no supe seguir la pista del buffer ya que es demasiado tedioso, IDM utiliza funciones optimizadas para realizar el trabajo con el buffer, ésto implica trabajo con bits y assembler puro y duro, en fin a penas pueda realizar el trabajo no dudare en escribir el cómo continuar descifrando ésta cadena.  
  
Sin embargo no he dejado ésto sin solución, mi solución ha sido la siguiente el buffer de recv() recibe el **HEADER** del server más la información de validación cifrada, todo ésto en un solo buffer.  
  
**HEADER:**  
  
**HTTP/1.1 200 OK**

**Date**: Tue, 18 Aug 2015 03:45:07 GMT

**Server**: Apache/2.4.6 (Red Hat Enterprise Linux) OpenSSL/1.0.1e-fips mod\_fcgid/2.3.9

**Accept-Ranges**: none

**Set-Cookie**: IDM=8e90f8f7.51d8dbe7d3180; path=/; expires=Thu, 26-Nov-15 03:45:07 GMT; domain=.internetdownloadmanager.com

**Content-Length**: 64

**Content-Type**: text/html; charset=UTF-8

Desde el header podemos obtener la siguiente información:  
  
1.- **Server** : Tenemos información acerca de qué servidor web y versión está utilizando, además de datos como el sistema operativo y versión de **OPENSSL**, ésta información sería necesaria por si quisiéramos explotar algún tipo de vulnerabilidad desde el lado del servidor.  
  
2.- **Set-Cookie** : El servidor nos ha devuelto una cookie, tal vez tenga algo que ver con la validación del serial.  
  
3.- **Content-Length** : Nos dice la cantidad de datos que se adjunta después del HEADER, nos dice 64bytes pero veremos que pasa lo mismo que con los buffer de send(), en ésta ocasión no se utilizan los 64bytes.  
  
  
**DATA**:  
  
Éstos son los 64bytes que nos devuelve el servidor.  
  
**\x**73 **\x**D5 **\x**CF **\x**B8 **\x**A4 **\x**07 **\x**89 **\x**80

**\x**31 **\x**E4 **\x**35 **\x**6B **\x**2A **\x**CA **\x**FE **\x**43

**\x**B6 **\x**1F **\x**81 **\x**1F **\x**5A **\x**1B **\x**4D **\x**36

**\x**46 **\x**8F **\x**3C **\x**F2 **\x**5C **\x**68 **\x**EE **\x**21

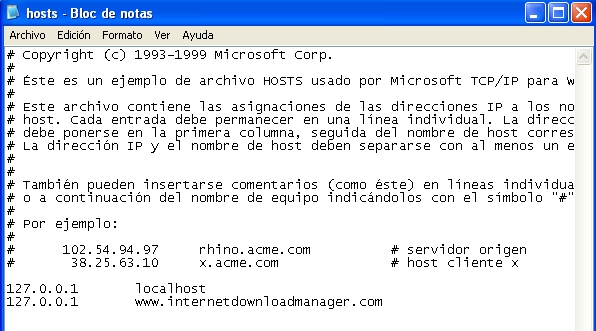
**\x**46 **\x**8F **\x**3C **\x**F2 **\x**5C **\x**68 **\x**EE **\x**21

**\x**46 **\x**8F **\x**3C **\x**F2 **\x**5C **\x**68 **\x**EE **\x**21

**\x**46 **\x**8F **\x**3C **\x**F2 **\x**5C **\x**68 **\x**EE **\x**21

**\x**46 **\x**8F **\x**3C **\x**F2 **\x**5C **\x**68 **\x**EE **\x**21

Si ponemos atención solo **24bytes**, vienen con información, si quisieramos descifrar la cadena tendríamos que descifrar esos 24bytes.  
  
**Solución**:

Mi solución a la validación por web es la siguiente:   
  
1.- Vamos al archivo HOSTS, ponemos los datos así.  
  
  
  
Nuestro IDM no dirá nada acerca del archivo hosts ya que se encuentra parchado y no encontrará la información del dominio.  
  
2.- Antes de ir a IDM y registrarnos ejecutamos el siguiente script en python.

Enlace en pastebin : http://pastebin.com/yVeycc7r

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

import socket

HEAD = "HTTP/1.1 200 OK" + "\r\n"

HEAD += "Date: Tue, 18 Aug 2015 03:45:07 GMT" + "\r\n"

HEAD += "Server: Apache/2.4.6 (Red Hat Enterprise Linux) OpenSSL/1.0.1e-fips mod\_fcgid/2.3.9" + "\r\n"

HEAD += "Accept-Ranges: none" + "\r\n"

HEAD += "Set-Cookie: IDM=8e90f8f7.51d8dbe7d3180; path=/; expires=Thu, 26-Nov-15 03:45:07 GMT; domain=.internetdownloadmanager.com" + "\r\n"

HEAD += "Content-Length: 64" + "\r\n"

HEAD += "Content-Type: text/html; charset=UTF-8" + "\r\n\r\n"

#ORIGINAL, NO SE ENVIA

DATA =""

#DATA += "\x73\xD5\xCF\xB8\xA4\x07\x89\x80"

#DATA += "\x31\xE4\x35\x6B\x2A\xCA\xFE\x43"

#DATA += "\xB6\x1F\x81\x1F\x5A\x1B\x4D\x36"

#DATA += "\x46\x8F\x3C\xF2\x5C\x68\xEE\x21"

#DATA += "\x46\x8F\x3C\xF2\x5C\x68\xEE\x21"

#DATA += "\x46\x8F\x3C\xF2\x5C\x68\xEE\x21"

#DATA += "\x46\x8F\x3C\xF2\x5C\x68\xEE\x21"

#DATA += "\x46\x8F\x3C\xF2\x5C\x68\xEE\x21"

#SE ENVIA LA SIGUIENTE DATA  
  
DATA = “\x41”\*64

s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind(('localhost',80))

s.listen(10)

a=s.accept()[0]

print "[ RECIBIENDO DATOS ]"

print a.recv(2048)

print a.recv(2048)

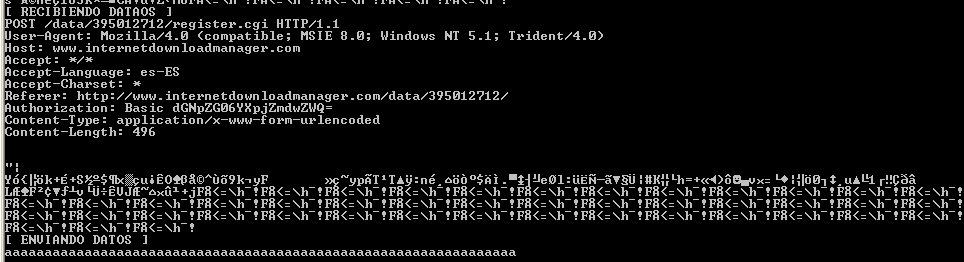
print "[ ENVIANDO DATOS ]"

print DATA

a.send(DATA)

print a.recv(2048)

s.close()

La idea del código en Python es que actúe como el servidor **WEB** que realiza la validación, a diferencia del servidor original, nosotros enviamos como DATA **64bytes** que contiene “A”, por lo tanto idm no los reconoce se cierra y cuando se abre nuevamente, es como si hubiésemos registrados.  
  
3.- Ejecutamos **IDM**, le rellenamos con datos falsos y enviará la petición a nuestro servidor en python.  
  
  
  
se cerrará, abrimos nuevamente el IDM y andará todo ok por un rato, puesto que para que funcione del todo bien, nuestro servidor debería correr 24x7.  
  
  
**Esto ha sido todo por ahora, espero que lo disfruten tanto como yo lo hice creando el tutorial, hasta la próxima ;) .**