Global Information Assurance Certification Paper

Creating a Baseline of Process Activity for Memory Forensics

GIAC (GCFA) Gold Certification

Author: Gordon Fraser, Gordon.fraser@ctipc.com
Advisor: Richard Carbone

Accepted: August 19th 2014

Abstract

A component of memory forensics is the examination of running processes looking for anomalies. However, this assumes that the analyst can recognize the anomalies. A frame of reference to assist the analyst is the creation of a baseline which identifies what is expected to be present in memory for a given configuration. The analyst can use the baseline as a reference to quickly filter out expected processes and to focus on what is not expected to be there. This paper covers the creation of a baseline for Windows Server 2008 R2 with several different configurations.

1. Introduction

SANS's Advanced Forensic Analysis and Incident Response course (Lee & Tilbury, 2013) defines a process for the examination of memory to identify indicators of compromise. It is comprised of the following six steps: identify rogue processes, analyze process objects, review network artifacts, look for evidence of code injection, check for signs of rootkits, and dump suspicious processes and drivers.

While identifying rogue processes the analyst is looking at what is running, parent processes, and when they started. Should a process be there? What is the parent process and is it what is expected? Did the process start when it was supposed to?

When analyzing process objects, the analyst examines the name of the executable, the executable's path, the command line parameters used for starting the process, and the security identifiers associated with it. Did the executable start from the directory from which it was expected? Are the command line parameters what we expected to see? Is the context in which the process is running what is expected? That is, was it run as System or was it run as a user? The analyst can also look at dynamic link libraries (DLLs) and kernel modules, which have been loaded into memory.

Next, the analyst examines the network connections that have been established and which process they are associated with. Are there any unusual network connections? Are the network connections associated with the process expected to have initiated them?

The common thread for each of the first three steps is looking for anything unusual. However, how does one know if something is unusual? Or, to turn the question around, how does one know what is expected? Searches through the literature and the web can provide some guidance, but it only provides a starting point. One solution is to establish a baseline of what is expected and use it as a reference of what to expect. Such a baseline could be used to filter out expected processes allowing the analyst to focus on the unexpected.

This paper starts out by creating a baseline from the literature, such as Windows Internals, 6th Edition Part 1 (Russinovich, Solomon & Ionescu, 2012a) and Part 2 (Russinovich, Solomon & Ionescu, 2012b), "Know your Windows Processes or Die

Trying" (Olsen, 2014), and The Art of Memory Forensics (Ligh, Case, Levy, and Walters, 2014). It then tests the baseline against a series of Windows 2008 R2 servers in order to validate and refine the baseline.

2. Creating and Validating a Windows Server 2008 R2 Baseline

2.1. Data Collection Approach

2.1.1. Memory Acquisition

The starting point for memory analysis is the acquisition of a memory image. There are a number of tools that can be used to acquire memory; *win64dd.exe* from MoonSols was used for this analysis. Best practice is to capture the memory image to an external device in order to minimize the impact the capture process has on the system being analyzed. Writing the image to a disk on the system, which is the subject of the investigation, could result in important data be overwritten and lost. Given the amount of memory being captured, this could amount to gigabytes of data. For this analysis, since we are only interested in the memory image and not the disk contents, memory was collected by running *win64dd.exe* from the *C:\tmp* directory on the hard drive of the server.

2.1.2. Memory Analysis

There are a number of tools that can be used to analyze the memory image. For the purpose of this analysis, Volatility was chosen. It is an open source memory analysis framework written in Python (Volatility Wiki, 2013).

Volatility requires the memory image format be defined when executing Volatility commands, except when analyzing a Windows XP Service Pack 2 memory image (Volatility Wiki, 2013). The format used in the analysis, Win2008R2SP1x64, was determined by running the command: `vol.py -f mem.img imageinfo`, where *mem.img* was the name of the image file being analyzed. An alternative to including the image format in every Volatility commands is to set the `VOLATILITY_PROFILE` environment variable using the command:

```
export VOLATILITY_PROFILE=Win2008R2SP1x64.
```

2.1.3. Server Installation

Windows Server 2008 R2 was installed in VMware Workstation 9.0 with VMware Tools installed. All variations of Windows were installed from the same installation ISO disk image. The Windows Server 2008 R2 version, obtained using the command: `systeminfo | find "OS"`, was 6.1.760 Service Pack 1 Build 7601. Updates to the operating system were not applied.

2.2. Establishing an Initial Baseline

As a starting point for constructing a baseline of processes in memory one needs a basic understanding of the core processes loaded by the operating system upon boot and the processes that are loaded when a user logs on. This has been documented in detail in Windows Internals, 6th Edition Part 1 (Russinovich, Solomon & Ionescu, 2012a) and Part 2 (Russinovich, Solomon & Ionescu, 2012b). A summary of the critical Windows processes can be found in the article "Know your Windows Processes or Die Trying" (Olsen, 2014), in The Art of Memory Forensics (Ligh, Case, Levy, and Walters, 2014), as well as on the SANS DFIR Digital Forensics and Incident Response Poster (Pilkington & Lee, 2014).

The first process that appears in the process list from memory is System. System is a container for kernel processes (Ligh, Case, Levy, and Walters, 2014). It has a static process ID of 4 and no parent process. System starts the session manager (*smss.exe*) (Olsen, 2014).

Smss.exe is the first user-mode process of the boot sequence. It is responsible for creating sessions. Two sessions are created on boot. Session 0 contains processes owned by the system and Windows services. Session 1 contains processes owned by the user. (Ligh, Case, Levy, and Walters, 2014) *Smss.exe* starts a copy of the client/server runtime subsystem (*csrss.exe*) for each session. It starts the Windows Initialization Process (*wininit.exe*) to initialize session 0 and the Windows Logon Process (*winlogon.exe*) to initialize user sessions. Each user logon has a unique session ID that is created when they log on. When creating a new session, *smss.exe* creates a child instance of itself

which initializes the session and then exits. For this reason, the parent process ID (PPID) of *csrss.exe*, *wininit.exe*, and *winlogon.exe* do not map back to the process ID (PID) of a process in memory (Pilkington & Lee, 2014).

Wininit.exe performs the user-mode initialization processes that run in session 0. These include Local Security Authority (*lsass.exe*), Load Session Manager Service (*lsm.exe*), and Service Control Manager (*services.exe*). Lsass.exe is responsible for the local security policy. *Lsm.exe* manages terminal server sessions, calling *smss.exe* when a new session needs to be started (Olsen, 2014).

Windows, like most operating systems, has processes that are not associated with a specific interactive user. Instead, they run independent of user logons. These are services and are started by *services.exe*. Windows does not run each service as its own process. Instead, it groups services together with common characteristics into service groups. These service groups are started using a generic process called *svchost.exe*. This is why multiple *svchost.exe* can be seen running in memory. *Svchost.exe* is launched with a *-k* parameter specifying which service group to start. We will populate the initial baseline with the major service groups defined in Windows Internals Part 1 (Russinovich, Solomon & Ionescu, 2012a) listed in Table 1.

Service Group	Owner
Local Service	Local Service
LocalServiceAndNoImpersonation	Local Service
LocalServiceNetworkRestricted	Local Service
LocalServiceNoNetwork	Local Service
LocalSystemNetworkRestricted	Local System
NetworkService	Network Service
NetworkServiceAndNoImpersonation	Network Service
NetworkServiceNetworkRestricted	Network Service

Table 1: Major Service Groupings

An initial baseline for process memory analysis can be built using this description of the Windows boot process. Table 2 summarizes the processes one would expect to find in a memory dump for session 0.

Name	Parent Process	Session	Event	Owner	Path
System	-	0	Boot	Local System	%SystemRoot%\System32
smss.exe	System	0	Boot	Local System	%SystemRoot%\System32
csrss.exe	smss.exe	0	Boot	Local System	%SystemRoot%\System32
wininit.exe	smss.exe	0	Boot	Local System	%SystemRoot%\System32
services.exe	wininit.exe	0	Boot	Local System	%SystemRoot%\System32
lsass.exe	wininit.exe	0	Boot	Local System	%SystemRoot%\System32
lsm.exe	wininit.exe	0	Boot	Local System	%SystemRoot%\System32
svchost.exe	services.exe	0	Boot		%SystemRoot%\System32

Table 2: Initial Process Baseline

Winlogon.exe is responsible for interactive logon processes. It launches the *LogonUI.exe* process which manages the user logon interaction, changing of passwords, and locking and unlocking the workstation (Russinovich, Solomon & Ionescu, 2012a). Once the user successfully authenticates, the shell process, as defined in the registry value *HKLM\SOFTWARE\Microsoft\Windows NT\Current Version\Winlogon\shell*, is started. The default shell process is *Explorer.exe*. This is started by the process *Userinit*, which ends once *explorer.exe* is started. Therefore, we should not find its parent process running. If the user's logon is via remote desktop, rather than the console, the process *rdpclip.exe* will also be found in memory associated with the user's session (Ligh, Case, Levy, and Walters, 2014).

There are three states that a user session can be in: user not logged in, user logged in from console, and user logged in remotely. Table 3 lists the processes that are expected to be seen for each of these states. It constitutes the initial baseline for user logon processes. The parent process is not listed because its parent exits and so the PPID does not track back to a process in memory. *Session n* refers to any user session since a user session could be session 1, session 2, etc.

Name	Parent Process	Session	Owner	Path
User not logged in				
csrss.exe		n	Local System	%SystemRoot%\System32
winlogon.exe		n	Local System	%SystemRoot%\System32
LogonUI.exe	winlogon.exe	n	Local System	%SystemRoot%\System32
User logged in on console				
csrss.exe		n	Local System	%SystemRoot%\System32
winlogon.exe		n	Local System	%SystemRoot%\System32
explorer.exe		n	User	%SystemRoot%
User logged in remotely				

Gordon Fraser, Gordon.fraser@ctipc.com

Name	Parent Process	Session	Owner	Path
csrss.exe		n	Local System	%SystemRoot%\System32
winlogon.exe		n	Local System	%SystemRoot%\System32
rdpclip.exe	services.exe	n	User	%SystemRoot%\System32
explorer.exe		n	User	%SystemRoot%

Table 3: Initial Process Baseline for logon

In addition to processes, there are three other artifacts of interest when creating a baseline for memory forensics -- dynamic link libraries, modules, and drivers. Dynamic link libraries (DLLs) contain code and resources that are shared between multiple processes. Modules are code that is loaded into the Operating System kernel. Finally, drivers are the code that allows the computer to communicate with hardware devices (Ligh, Case, Levy, and Walters, 2014). Since each of these artifacts consists of a large number of objects, the baseline will be created from the analysis of live systems rather than from literature searches.

Another set of attributes of interest to the forensic analyst is the network connections that have been opened up by processes. These can be viewed using the *netscan* plugin. The network connections portion of the baseline will be built by gathering data from live systems and then analyzing it.

2.3. Testing Baseline against Generic Windows Server 2008 R2

Now that we have an initial memory baseline, we can compare it against a memory image collected from a live system in order to validate and improve it. The first Windows 2008 R2 server was built using a generic standard edition installation with the server being part of a workgroup. Since this is a fresh installation from Windows media, it is assumed that it is "clean".

2.3.1. Identify Rogue Processes

Following the memory analysis methodology, we will start with Step 1: looking for rogue processes. Since there should be no rogue process, we are really looking for processes that are not part of our initial baseline.

The output below provides a process listing using Volatility's *pslist* plugin. In order to fit it on the page the cut command was used to remove some irrelevant output.

To separate the boot process from the logon processes, there was a gap in time between them. This is evident from the start time column, seen below.

```
$ vol.py -f win2008r2-01-a.img pslist | cut -c 20-76,84-104
Volatility Foundation Volatility Framework 2.3.1
Name                    PID   PPID   Thds     Hnds   Sess Start
-------------------- ------ ------ ------ -------- ------ -------------------
System                     4      0     76      460 ------ 2014-06-22 13:26:36
smss.exe                 232      4      2       29 ------ 2014-06-22 13:26:36
csrss.exe                324    316      9      335      0 2014-06-22 13:26:42
csrss.exe                376    368     10      189      1 2014-06-22 13:26:43
wininit.exe              384    316      3       79      0 2014-06-22 13:26:43
winlogon.exe             420    368      3       96      1 2014-06-22 13:26:43
services.exe             480    384      8      189      0 2014-06-22 13:26:44
lsass.exe                488    384      6      530      0 2014-06-22 13:26:45
lsm.exe                  496    384     10      145      0 2014-06-22 13:26:45
svchost.exe              588    480     11      345      0 2014-06-22 13:26:48
svchost.exe              656    480      6      233      0 2014-06-22 13:26:49
svchost.exe              744    480     13      288      0 2014-06-22 13:26:49
svchost.exe              788    480     26      824      0 2014-06-22 13:26:49
svchost.exe              836    480     10      508      0 2014-06-22 13:26:50
svchost.exe              884    480      7      198      0 2014-06-22 13:26:51
svchost.exe              928    480     16      429      0 2014-06-22 13:26:51
svchost.exe              216    480     17      289      0 2014-06-22 13:26:53
spoolsv.exe              904    480     13      313      0 2014-06-22 13:26:54
svchost.exe             1040    480      3       46      0 2014-06-22 13:26:56
vmtoolsd.exe            1096    480      9      253      0 2014-06-22 13:26:56
TPAutoConnSvc.          1292    480     10      140      0 2014-06-22 13:26:59
dllhost.exe             1456    480     13      194      0 2014-06-22 13:27:01
msdtc.exe               1612    480     12      147      0 2014-06-22 13:27:03
svchost.exe             1752    480      5       67      0 2014-06-22 13:29:01
taskhost.exe            1276    480      5      118      1 2014-06-22 13:34:03
TPAutoConnect.          1328   1292      5      126      1 2014-06-22 13:34:03
conhost.exe             1556    376      1       30      1 2014-06-22 13:34:03
dwm.exe                 1116    884      3       65      1 2014-06-22 13:34:03
explorer.exe             848    968     15      478      1 2014-06-22 13:34:03
vmtoolsd.exe            2012    848      7      184      1 2014-06-22 13:34:06
cmd.exe                  332    848      1       20      1 2014-06-22 13:39:31
conhost.exe             1228    376      2       36      1 2014-06-22 13:39:31
win64dd.exe              868    332      2       49      1 2014-06-22 13:52:33
```

The bolded lines above identify those processes that are not already present in the baseline. These ones need to be examined more closely.

The servers are running on VMware Workstation, so there may be processes associated with VMware that are not part of the normal generic Windows installation. Indeed, we find three processes -- *TPAutoConnSvc.exe*, *TPAutoConnect.exe*, and *vmtoolsd.exe*. A clue to this association to VMware is given when examining the process information in more detail. Each of these processes is being run out of the directory *C:\Program Files\VMware\VMware Tools* as shown in the output below from the *dlllist* plugin.

Gordon Fraser, Gordon.fraser@ctipc.com

```
$ vol.py -f win2008r2-01-a.img dlllist -p 1096 | grep -B 1 -i "command line"
Volatility Foundation Volatility Framework 2.3.1
vmtoolsd.exe pid:   1096
Command line : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe"

$ vol.py -f win2008r2-01-a.img dlllist -p 1292 | grep -B 1 -i "command line"
Volatility Foundation Volatility Framework 2.3.1
TPAutoConnSvc. pid:   1292
Command line : "C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe"

$ vol.py -f win2008r2-01-a.img dlllist -p 1328 | grep -B 1 -i "command line"
Volatility Foundation Volatility Framework 2.3.1
TPAutoConnect. pid:   1328
Command line : TPAutoConnect.exe -q -i vmware -a COM1 -F 30
```

The VMware Knowledge base (2014) identifies *TPAutoConnect.exe* and *TPAutoConnSvc.exe* as being started when the Virtual Printing module is installed. We can see from the pslist output that *TPAutoConnSvc.exe* is started by the services control manager (SCM) in session 0. *TPAutoConnect.exe* is started as part of the initialization of session 1 by *TPAutoConnSvc.exe*. The VMware tools service (*vmtoolsd.exe*) is installed on guest Windows operating systems (VMware, 2011). From the output of *pslist*, we see that there is an instance of *vmtoolsd.exe* running in both session 0 and session 1. Thus, these processes are added to the baseline.

Some memory artifacts are introduced by a user logon. *LogonUI.exe* is not present because a user is logged in. Instead, several processes associated with a successful logon are present including: *taskhost.exe*, *conhost.exe*, and *dwm.exe*. These processes where not part of the initial baseline. Table 4 provides a revised list of the processes that are started when a user logs on.

Name	Parent Process	Session	Owner	Path
User not logged in				
csrss.exe		n	Local System	%SystemRoot%\System32
winlogon.exe		n	Local System	%SystemRoot%\System32
LogonUI.exe	winlogon.exe	n	Local System	%SystemRoot%\System32
User logged in on console				
csrss.exe		n	Local System	%SystemRoot%\System32
winlogon.exe		n	Local System	%SystemRoot%\System32
taskhost.exe	services.exe	n	user	%SystemRoot%\System32
TPAutoConnect.exe	TPAutoConnSvc.exe	n	user	
conhost.exe	crsrr.exe	n	user	%SystemRoot%\System32

Name	Parent Process	Session	Owner	Path
dwm.exe	svchost.exe	n	user	%SystemRoot%\System32
explorer.exe		n	user	%SystemRoot%
vmtoolsd.exe	explorer.exe	n	user	C:\Program Files\VMware\VMware Tools\
User logged in remotely				
csrss.exe		n	Local System	%SystemRoot%\System32
winlogon.exe		n	Local System	%SystemRoot%\System32
taskhost.exe	services.exe	n	user	%SystemRoot%\System32
TPAutoConnect.exe	TPAutoConnSvc.exe	n	user	C:\Program Files\VMware\VMware Tools\
conhost.exe	crsrr.exe	n	user	%SystemRoot%\System32
dwm.exe	svchost.exe	n	user	%SystemRoot%\System32
rdpclip.exe	services.exe	n	user	%SystemRoot%\System32
explorer.exe		n	user	%SystemRoot%
vmtoolsd.exe	explorer.exe	n	user	C:\Program Files\VMware\VMware Tools\

Table 4: Revised Process Baseline for logon

Another group of artifacts is associated with an actual memory acquisition. *Cmd.exe* was run in administrative mode (PID - 332) to get a command window from which *win64dd.exe* (PID 868) was started to capture the memory image. When *cmd.exe* is run, Windows also starts up a supporting process, console host (*conhost.exe* - PID 1228). While these last processes are not part of the process baseline, they are taken into account when analyzing the memory image.

We also see three more processes: *spoolsvc.exe*, *dllhost.exe*, and *msdtc.exe*, which were started by *services.exe* that are not part of the initial baseline. *Dllhost.exe* is associated with the management of Component Object Model (COM) objects (Startup Programs Database, 2014c). *Msdtc.exe* is the Distributed Transaction Coordinator (Startup Programs Database, 2014d). *Spoolsvc.exe* is the spooler service and is associated with printing (Startup Programs Database, 2014e).

2.3.2. Analyze Process Objects

Step 2 in the memory analysis methodology is to examine process objects. We will use several Volatility plugins to do this starting with the *dlllist* plugin. Since it is

very verbose, we ran the command limiting its output to one process, *smss.exe*, which has a PID of 232. The output of this command provides information about the command used to initiate the process as well as the DLLs associated with the process, as seen below.

```
$ vol.py -f win2008r2-01-a.img -p 232 dlllist
Volatility Foundation Volatility Framework 2.3.1
************************************************************************
smss.exe pid:    232
Command line : \SystemRoot\System32\smss.exe


Base                             Size      LoadCount Path
------------------ -------------- -------------- ----
0x0000000047a50000        0x20000         0xffff \SystemRoot\System32\smss.exe
0x0000000077120000       0x1a9000         0xffff C:\Windows\SYSTEM32\ntdll.dll
```

Since what we are interested in are the names of the process, the process ID, and the command line used to start the process, we can use grep to create an abbreviated output from the *dlllist* plugin that only prints out the lines with the PID and the command line, as seen below.

```
$ vol.py -f win2008r2-01-a.img   dlllist | grep -B 1 -i "command line"
Volatility Foundation Volatility Framework 2.3.1
smss.exe pid:    232
Command line : \SystemRoot\System32\smss.exe
--
csrss.exe pid:    324
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,20480,768 Windows=On SubSystemType=Windows
ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4
ProfileControl=Off MaxRequestThreads=16
--
csrss.exe pid:    376
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,20480,768 Windows=On SubSystemType=Windows
ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4
ProfileControl=Off MaxRequestThreads=16
--
wininit.exe pid:    384
Command line : wininit.exe
--
winlogon.exe pid:    420
Command line : winlogon.exe
--
services.exe pid:    480
Command line : C:\Windows\system32\services.exe
--
lsass.exe pid:    488
Command line : C:\Windows\system32\lsass.exe
--
lsm.exe pid:    496
Command line : C:\Windows\system32\lsm.exe
--
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
svchost.exe pid:    588
Command line : C:\Windows\system32\svchost.exe -k DcomLaunch
--
svchost.exe pid:    656
Command line : C:\Windows\system32\svchost.exe -k RPCSS
--
svchost.exe pid:    744
Command line : C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted
--
svchost.exe pid:    788
Command line : C:\Windows\system32\svchost.exe -k netsvcs
--
svchost.exe pid:    836
Command line : C:\Windows\system32\svchost.exe -k LocalService
--
svchost.exe pid:    884
Command line : C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted
--
svchost.exe pid:    928
Command line : C:\Windows\system32\svchost.exe -k NetworkService
--
svchost.exe pid:    216
Command line : C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork
--
spoolsv.exe pid:    904
Command line : C:\Windows\System32\spoolsv.exe
--
svchost.exe pid:   1040
Command line : C:\Windows\system32\svchost.exe -k regsvc
--
vmtoolsd.exe pid:   1096
Command line : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe"
--
TPAutoConnSvc. pid:   1292
Command line : "C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe"
--
dllhost.exe pid:   1456
Command line : C:\Windows\system32\dllhost.exe /Processid:{02D4B3F1-FD88-11D1-
960D-00805FC79235}
--
msdtc.exe pid:   1612
Command line : C:\Windows\System32\msdtc.exe
--
svchost.exe pid:   1752
Command line : C:\Windows\system32\svchost.exe -k
LocalServiceAndNoImpersonation
--
taskhost.exe pid:   1276
Command line : "taskhost.exe"
--
TPAutoConnect. pid:   1328
Command line : TPAutoConnect.exe -q -i vmware -a COM1 -F 30
--
conhost.exe pid:   1556
Command line : \??\C:\Windows\system32\conhost.exe
--
dwm.exe pid:   1116
Command line : "C:\Windows\system32\Dwm.exe"
--
explorer.exe pid:    848
Command line : C:\Windows\Explorer.EXE
--
vmtoolsd.exe pid:   2012
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
Command line : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
--
cmd.exe pid:    332
Command line : "C:\Windows\system32\cmd.exe"
--
conhost.exe pid:   1228
Command line : \??\C:\Windows\system32\conhost.exe
--
win64dd.exe pid:    868
Command line : win64dd  /f win2008r2-01-a.img
```

We compare the output of the *dlllist* plugin to our baseline to validate the commands used to start processes and to identify the service groups started by *services.exe*. In doing so, we quickly identify three service groups: *DcomLaunch*, *RPCSS*, and *regsvc*; all of which are not in the baseline and need to be added.

The process analysis step includes checking to make sure the process is running under the expected account. This information is available using the *getsid* Volatility plugin. The example below shows the output for one process with a PID of 488.

```
$ vol.py -f win2008r2-01-a.img -p 488 getsids
Volatility Foundation Volatility Framework 2.3.1
lsass.exe (488): S-1-5-18 (Local System)
lsass.exe (488): S-1-5-32-544 (Administrators)
lsass.exe (488): S-1-1-0 (Everyone)
lsass.exe (488): S-1-5-11 (Authenticated Users)
lsass.exe (488): S-1-16-16384 (System Mandatory Level)
```

The Volatility output can be shortened using the *uniq* command as shown below. Here is a list of each process and the account it runs under. This technique of using *uniq* is not perfect. A few duplicate lines show up for System, but it still simplifies the analysis process.

```
$ vol.py -f win2008r2-01-a.img getsids | uniq -w 18
Volatility Foundation Volatility Framework 2.3.1
System (4): S-1-5-18 (Local System)
System (4): S-1-1-0 (Everyone)
System (4): S-1-5-11 (Authenticated Users)
System (4): S-1-16-16384 (System Mandatory Level)
smss.exe (232): S-1-5-18 (Local System)
csrss.exe (324): S-1-5-18 (Local System)
csrss.exe (376): S-1-5-18 (Local System)
wininit.exe (384): S-1-5-18 (Local System)
winlogon.exe (420): S-1-5-18 (Local System)
services.exe (480): S-1-5-18 (Local System)
lsass.exe (488): S-1-5-18 (Local System)
lsm.exe (496): S-1-5-18 (Local System)
svchost.exe (588): S-1-5-18 (Local System)
svchost.exe (656): S-1-5-20 (NT Authority)
svchost.exe (744): S-1-5-19 (NT Authority)
svchost.exe (788): S-1-5-18 (Local System)
svchost.exe (836): S-1-5-19 (NT Authority)
svchost.exe (884): S-1-5-18 (Local System)
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
svchost.exe (928): S-1-5-20 (NT Authority)
svchost.exe (216): S-1-5-19 (NT Authority)
spoolsv.exe (904): S-1-5-18 (Local System)
svchost.exe (1040): S-1-5-19 (NT Authority)
vmtoolsd.exe (1096): S-1-5-18 (Local System)
TPAutoConnSvc. (1292): S-1-5-18 (Local System)
dllhost.exe (1456): S-1-5-18 (Local System)
msdtc.exe (1612): S-1-5-20 (NT Authority)
svchost.exe (1752): S-1-5-19 (NT Authority)
taskhost.exe (1276): S-1-5-21-2236604341-3981238657-2714753860-1000
TPAutoConnect. (1328): S-1-5-21-2236604341-3981238657-2714753860-1000
conhost.exe (1556): S-1-5-21-2236604341-3981238657-2714753860-1000
dwm.exe (1116): S-1-5-21-2236604341-3981238657-2714753860-1000
explorer.exe (848): S-1-5-21-2236604341-3981238657-2714753860-1000
vmtoolsd.exe (2012): S-1-5-21-2236604341-3981238657-2714753860-1000
cmd.exe (332): S-1-5-21-2236604341-3981238657-2714753860-1000
conhost.exe (1228): S-1-5-21-2236604341-3981238657-2714753860-1000
win64dd.exe (868): S-1-5-21-2236604341-3981238657-2714753860-1000
```

One item of note concerning the above output is that two SIDs are listed by Volatility as NT Authority. A better translation, more in line with the literature, is S-1-5-19 translates to Local Service and S-1-5-20 translates to Network Service (Microsoft Knowledge Base, 2013a). The SIDs with the long list of numbers ending in "-1000" are associated with a user. Thus, processes such as *taskhost.exe*, *cmd.exe*, and *win64dd.exe* are running in the user's context.

As expected, different services started using *svchost* have different SIDs based on the requirements of the service group. Moreover, processes associated with the logged on user are associated with a user SID.

Another memory artifact to analyze is the DLLs. A complete list of DLLs generated by the *dlllist* plugin is rather lengthy. As seen earlier, it also includes other information in addition to the DLLs. We can take advantage of the fact that each line listing a DLL begins with a "0x" and produce output from the *dlllist* plugin which only lists the DLLs. The following command lists those lines containing a DLL and then counts them. In all, 1,411 DLLs were identified.

```
$ vol.py -f win2008r2-01-a.img dlllist | grep "^0x" | cut -c 20-37,57- | wc -l
Volatility Foundation Volatility Framework 2.3.1
1411
```

By examining the list of DLLs, it becomes clear very quickly that there are many duplicates which exist because many different processes use the same DLL. By sorting the list and then only displaying unique lines and ignoring case, we can trim the list down

considerably. The command below shows a reduction to only 370 lines from the memory image.

```
$ vol.py -f win2008r2-01-a.img dlllist | grep "^0x" | cut -c 20-37,57- | sort |
uniq -i | wc -l
Volatility Foundation Volatility Framework 2.3.1
370
```

For our baseline, rather than creating a table listing 370 items, we created a file that can then be compared to the output of the same command run against another memory image to look for DLLs not listed in the baseline. The file should be reviewed to make sure that there are no entries that do not belong. One entry, *win64dd.exe*, should be removed since it was introduced as part of the memory capture process. The command used to create the DLL baseline is:

```
$ vol.py -f win2008r2-01-a.img dlllist | grep "^0x" | cut -c 20-37,57- | sort |
uniq -i > dll-baseline-01.lst
Volatility Foundation Volatility Framework 2.3.
```

Similarly, Volatility provides a plugin to list kernel modules in memory called *modules*. Reviewing the output reveals 147 modules in memory. A similar approach was used to create a module baseline as shown below.

```
$ vol.py -f win2008r2-01-a.img modules | cut -c 20-40,60- | grep "0x" | sort |
uniq -i | wc -l
Volatility Foundation Volatility Framework 2.3.1
147

$ vol.py -f win2008r2-01-a.img modules | cut -c 20-40,60- | grep "0x" | sort |
uniq -i > module-base-01.lst
Volatility Foundation Volatility Framework 2.3.1

$ cat module-base-01.lst | wc -l
147
```

The file should be reviewed to make sure that there are no entries that do not belong. *Win64dd.sys* was removed since it is an artifact of the memory capture process.

2.3.3. Review Network Artifacts

Starting with Windows Vista and Windows Server 2008 Microsoft changed the dynamic port range assignment from 1025 through 5000 to 49152 through 65535, as per the IANA recommendation (Microsoft Knowledge Base, 2013c). The Volatility *netscan* plugin provides a list of open ports and which process owns it. In the case of each

svchost.exe instance, further analysis needs to be done to determine which service group owns it. For example, consider the following:

```
$ vol.py -f win2008r2-01-a.img netscan | cut -c 12-18,21-40,51-63,88-92,94-112
| uniq -w 20
Volatility Foundation Volatility Framework 2.3.1
Proto  Local Address        Foreign Addr  Pid     Owner
TCPv4  0.0.0.0:49156        0.0.0.0:0     480     services.exe
TCPv6  :::49156             :::0          480     services.exe
TCPv4  0.0.0.0:445          0.0.0.0:0     4       System
TCPv6  :::445               :::0          4       System
TCPv4  0.0.0.0:47001        0.0.0.0:0     4       System
TCPv6  :::47001             :::0          4       System
TCPv4  0.0.0.0:49153        0.0.0.0:0     744     svchost.exe
TCPv6  :::49153             :::0          744     svchost.exe
TCPv4  0.0.0.0:49154        0.0.0.0:0     788     svchost.exe
TCPv4  192.168.139.129:139  0.0.0.0:0     4       System
TCPv4  0.0.0.0:135          0.0.0.0:0     656     svchost.exe
TCPv6  :::135               :::0          656     svchost.exe
TCPv4  0.0.0.0:49152        0.0.0.0:0     384     wininit.exe
TCPv6  :::49152             :::0          384     wininit.exe
TCPv4  0.0.0.0:49155        0.0.0.0:0     488     lsass.exe
TCPv6  :::49155             :::0          488     lsass.exe
TCPv4  0.0.0.0:49154        0.0.0.0:0     788     svchost.exe
TCPv6  :::49154             :::0          788     svchost.exe
UDPv4  0.0.0.0:5355         *:*           928     svchost.exe
UDPv4  0.0.0.0:0            *:*           928     svchost.exe
UDPv6  :::0                 *:*           928     svchost.exe
UDPv4  0.0.0.0:123          *:*           836     svchost.exe
UDPv6  :::123               *:*           836     svchost.exe
UDPv4  0.0.0.0:123          *:*           836     svchost.exe
UDPv4  0.0.0.0:0            *:*           836     svchost.exe
UDPv6  :::0                 *:*           836     svchost.exe
UDPv4  0.0.0.0:0            *:*           836     svchost.exe
UDPv4  192.168.139.129:137  *:*           4       System
UDPv4  0.0.0.0:5355         *:*           928     svchost.exe
UDPv6  :::5355              *:*           928     svchost.exe
```

Table 5 provides a list of ports opened by Windows Server 2008 R2 as identified by the *netscan* plugin. This list establishes the initial baseline and includes some services that an organization might disable for security reasons including Server Message Block (SMB) and NETBIOS.

Port	Protocol	Service/Process	
123	udp	svchost.exe (LocalService)	Windows Time Services
135	tcp	RPCSS	Terminal Services
137	udp	System	NETBIOS Name Resolution
138	tcp	System	NETBIOS Datagram Service
139	tcp	System	NETBIOS Session Service
445	tcp	System	SMB

Port	Protocol	Service/Process	

Gordon Fraser, Gordon.fraser@ctipc.com

5355	udp	svchost.exe (NetworkService)	Local Link Multicast Name Resolution (LLMNR)
47001	tcp	System	Windows Remote Management listener (WINRM)
49152	tcp	wininit.exe	
49153	tcp	svchost.exe (LocalServiceNetworkRestricted)	
49154	tcp	svchost.exe (NetworkService)	
49155	tcp	lsass.exe	
49156	tcp	services.exe	

Table 5: Initial Baseline of open ports

2.4. Testing the Baseline against a Domain Attached Windows Server 2008 R2

We repeat the process again using the new revised baseline against another system to further validate and refine it. The second Windows Server 2008 R2 server for testing the baseline against was built using the same configuration as the first with the exception that is was attached to a domain and remote logon was enabled to see what differences exist as a result. In this image two users have logged into the server, one remotely, using Microsoft Terminal Server Connection (*mstsc*) and one via the console. The memory image was created by the remote user.

2.4.1. Identify Rogue Processes

Our analysis starts by running the *pslist* plugin to get a listing of processes so we can identify rogue processes. Consider the following:

```
$ vol.py -f win2008r2-03-s1.img pslist | cut -c 20-53,70-76,84-114
Volatility Foundation Volatility Framework 2.3.1
Name                 PID   PPID   Sess Start
-------------------- ------ ------ ------ ------------------------------
System                 4      0 ------ 2014-07-28 01:26:59 UTC+0000
smss.exe             224      4 ------ 2014-07-28 01:26:59 UTC+0000
csrss.exe            316    308      0 2014-07-28 01:27:04 UTC+0000
wininit.exe          368    308      0 2014-07-28 01:27:04 UTC+0000
services.exe         472    368      0 2014-07-28 01:27:05 UTC+0000
lsass.exe            480    368      0 2014-07-28 01:27:05 UTC+0000
lsm.exe              488    368      0 2014-07-28 01:27:05 UTC+0000
svchost.exe          584    472      0 2014-07-28 01:27:10 UTC+0000
svchost.exe          660    472      0 2014-07-28 01:27:11 UTC+0000
svchost.exe          740    472      0 2014-07-28 01:27:11 UTC+0000
svchost.exe          796    472      0 2014-07-28 01:27:11 UTC+0000
svchost.exe          848    472      0 2014-07-28 01:27:12 UTC+0000
svchost.exe          888    472      0 2014-07-28 01:27:12 UTC+0000
svchost.exe          932    472      0 2014-07-28 01:27:13 UTC+0000
svchost.exe          236    472      0 2014-07-28 01:27:14 UTC+0000
```

```
spoolsv.exe              324    472      0 2014-07-28 01:27:15 UTC+0000
svchost.exe             1092    472      0 2014-07-28 01:27:16 UTC+0000
vmtoolsd.exe            1148    472      0 2014-07-28 01:27:16 UTC+0000
svchost.exe             1412    472      0 2014-07-28 01:27:18 UTC+0000
svchost.exe             1472    472      0 2014-07-28 01:27:18 UTC+0000
TPAutoConnSvc.          1508    472      0 2014-07-28 01:27:19 UTC+0000
dllhost.exe             1776    472      0 2014-07-28 01:27:22 UTC+0000
msdtc.exe               1896    472      0 2014-07-28 01:27:23 UTC+0000
svchost.exe             1204    472      0 2014-07-28 01:29:19 UTC+0000
sppsvc.exe              1068    472      0 2014-08-02 20:10:58 UTC+0000
TrustedInstall           252    472      0 2014-08-02 20:11:00 UTC+0000
csrss.exe               3020   2956      1 2014-08-02 20:16:53 UTC+0000
winlogon.exe            2896   2956      1 2014-08-02 20:16:53 UTC+0000
taskhost.exe            1264    472      1 2014-08-02 20:17:25 UTC+0000
dwm.exe                 2328    888      1 2014-08-02 20:17:25 UTC+0000
explorer.exe            1284   3068      1 2014-08-02 20:17:25 UTC+0000
vmtoolsd.exe            1556   1284      1 2014-08-02 20:17:25 UTC+0000
TPAutoConnect.          2712   1508      1 2014-08-02 20:17:25 UTC+0000
conhost.exe             2856   3020      1 2014-08-02 20:17:25 UTC+0000
csrss.exe               2232    376      2 2014-08-02 20:18:27 UTC+0000
winlogon.exe            2124    376      2 2014-08-02 20:18:27 UTC+0000
taskhost.exe            2116    472      2 2014-08-02 20:18:29 UTC+0000
rdpclip.exe             2108   1412      2 2014-08-02 20:18:29 UTC+0000
dwm.exe                 2088    888      2 2014-08-02 20:18:29 UTC+0000
explorer.exe            2708   1388      2 2014-08-02 20:18:29 UTC+0000
vmtoolsd.exe            1124   2708      2 2014-08-02 20:18:29 UTC+0000
TPAutoConnect.          2012   1508      2 2014-08-02 20:18:30 UTC+0000
conhost.exe              672   2232      2 2014-08-02 20:18:30 UTC+0000
cmd.exe                 1268   2708      2 2014-08-02 20:18:40 UTC+0000
conhost.exe             2904   2232      2 2014-08-02 20:18:40 UTC+0000
win64dd.exe             2836   1268      2 2014-08-02 20:19:53 UTC+0000
```

Two new processes appear in session 0 -- *sppsvc.exe* and *TrustedInstall* -- both of which were started by *services.exe*. *Sppsvc.exe* is Microsoft's software protection service and is associated with managing digital licenses for Windows and Microsoft applications (Startup Programs Database, 2014a). *TrustedInstaller.exe* is the Windows Modules Installer and is associated with Windows Updates (Startup Programs Database, 2014b). These processes may not have shown up in our initial baseline because it was created soon after system boot, while they were started later.

In examining the two user sessions 1 and 2, we see that they are consistent with the baseline.

2.4.2. Analyze Process Objects

Once again, we start the analysis of the process objects by running the *dlllist* plugin.

```
$ vol.py -f win2008r2-03-s1.img   dlllist | grep -B 1 -i "command line"
Volatility Foundation Volatility Framework 2.3.1
smss.exe pid:    224
Command line : \SystemRoot\System32\smss.exe
```

```
--
csrss.exe pid:    316
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,20480,768 Windows=On SubSystemType=Windows
ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4
ProfileControl=Off MaxRequestThreads=16
--
wininit.exe pid:    368
Command line : wininit.exe
--
services.exe pid:    472
Command line : C:\Windows\system32\services.exe
--
lsass.exe pid:    480
Command line : C:\Windows\system32\lsass.exe
--
lsm.exe pid:    488
Command line : C:\Windows\system32\lsm.exe
--
svchost.exe pid:    584
Command line : C:\Windows\system32\svchost.exe -k DcomLaunch
--
svchost.exe pid:    660
Command line : C:\Windows\system32\svchost.exe -k RPCSS
--
svchost.exe pid:    740
Command line : C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted
--
svchost.exe pid:    796
Command line : C:\Windows\system32\svchost.exe -k netsvcs
--
svchost.exe pid:    848
Command line : C:\Windows\system32\svchost.exe -k LocalService
--
svchost.exe pid:    888
Command line : C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted
--
svchost.exe pid:    932
Command line : C:\Windows\system32\svchost.exe -k NetworkService
--
svchost.exe pid:    236
Command line : C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork
--
spoolsv.exe pid:    324
Command line : C:\Windows\System32\spoolsv.exe
--
svchost.exe pid:   1092
Command line : C:\Windows\system32\svchost.exe -k regsvc
--
vmtoolsd.exe pid:   1148
Command line : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe"
--
svchost.exe pid:   1412
Command line : C:\Windows\System32\svchost.exe -k termsvcs
--
svchost.exe pid:   1472
Command line : C:\Windows\system32\svchost.exe -k
NetworkServiceNetworkRestricted
--
TPAutoConnSvc. pid:   1508
Command line : "C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe"
--
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
dllhost.exe pid:   1776
Command line : C:\Windows\system32\dllhost.exe /Processid:{02D4B3F1-FD88-11D1-
960D-00805FC79235}
--
msdtc.exe pid:   1896
Command line : C:\Windows\System32\msdtc.exe
--
svchost.exe pid:   1204
Command line : C:\Windows\system32\svchost.exe -k
LocalServiceAndNoImpersonation
--
sppsvc.exe pid:   1068
Command line : C:\Windows\system32\sppsvc.exe
--
TrustedInstall pid:    252
Command line : C:\Windows\servicing\TrustedInstaller.exe
--
csrss.exe pid:   3020
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,20480,768 Windows=On SubSystemType=Windows
ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4
ProfileControl=Off MaxRequestThreads=16
--
winlogon.exe pid:   2896
Command line : winlogon.exe
--
taskhost.exe pid:   1264
Command line : "taskhost.exe"
--
dwm.exe pid:   2328
Command line : "C:\Windows\system32\Dwm.exe"
--
explorer.exe pid:   1284
Command line : C:\Windows\Explorer.EXE
--
vmtoolsd.exe pid:   1556
Command line : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
--
TPAutoConnect. pid:   2712
Command line : TPAutoConnect.exe -q -i vmware -a COM1 -F 30
--
conhost.exe pid:   2856
Command line : \??\C:\Windows\system32\conhost.exe
--
csrss.exe pid:   2232
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
SharedSection=1024,20480,768 Windows=On SubSystemType=Windows
ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4
ProfileControl=Off MaxRequestThreads=16
--
winlogon.exe pid:   2124
Command line : winlogon.exe
--
taskhost.exe pid:   2116
Command line : "taskhost.exe"
--
rdpclip.exe pid:   2108
Command line : rdpclip
--
dwm.exe pid:   2088
Command line : "C:\Windows\system32\Dwm.exe"
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
--
explorer.exe pid:   2708
Command line : C:\Windows\Explorer.EXE
--
vmtoolsd.exe pid:   1124
Command line : "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
--
TPAutoConnect. pid:   2012
Command line : TPAutoConnect.exe -q -i vmware -a COM1 -F 30
--
conhost.exe pid:    672
Command line : \??\C:\Windows\system32\conhost.exe
--
cmd.exe pid:   1268
Command line : "C:\Windows\system32\cmd.exe"
--
conhost.exe pid:   2904
Command line : \??\C:\Windows\system32\conhost.exe
--
win64dd.exe pid:   2836
Command line : win64dd  /f win2008r2-03-s1.img
```

Two new services appear in the memory image: *termsvcs* and service group *NetworkServiceNetworkRestricted*. These would be expected upon having enabling remote desktop. All of the other processes are in our baseline and the paths for the executables are correct.

Running the *getsids* plugin provides the account under which the processes are running. *Termsvcs* and service group *NetworkServiceNetworkRestricted* are running under network service. All other processes are running under the expected accounts.

```
$ vol.py -f win2008r2-03-s1.img getsids | uniq -w 18
Volatility Foundation Volatility Framework 2.3.1
System (4): S-1-5-18 (Local System)
System (4): S-1-1-0 (Everyone)
System (4): S-1-5-11 (Authenticated Users)
System (4): S-1-16-16384 (System Mandatory Level)
smss.exe (224): S-1-5-18 (Local System)
csrss.exe (316): S-1-5-18 (Local System)
wininit.exe (368): S-1-5-18 (Local System)
services.exe (472): S-1-5-18 (Local System)
lsass.exe (480): S-1-5-18 (Local System)
lsm.exe (488): S-1-5-18 (Local System)
svchost.exe (584): S-1-5-18 (Local System)
svchost.exe (660): S-1-5-20 (NT Authority)
svchost.exe (740): S-1-5-19 (NT Authority)
svchost.exe (796): S-1-5-18 (Local System)
svchost.exe (848): S-1-5-19 (NT Authority)
svchost.exe (888): S-1-5-18 (Local System)
svchost.exe (932): S-1-5-20 (NT Authority)
svchost.exe (236): S-1-5-19 (NT Authority)
spoolsv.exe (324): S-1-5-18 (Local System)
svchost.exe (1092): S-1-5-19 (NT Authority)
vmtoolsd.exe (1148): S-1-5-18 (Local System)
svchost.exe (1412): S-1-5-20 (NT Authority)
svchost.exe (1472): S-1-5-20 (NT Authority)
TPAutoConnSvc. (1508): S-1-5-18 (Local System)
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
dllhost.exe (1776): S-1-5-18 (Local System)
msdtc.exe (1896): S-1-5-20 (NT Authority)
svchost.exe (1204): S-1-5-19 (NT Authority)
sppsvc.exe (1068): S-1-5-20 (NT Authority)
TrustedInstall (252): S-1-5-18 (Local System)
csrss.exe (3020): S-1-5-18 (Local System)
winlogon.exe (2896): S-1-5-18 (Local System)
taskhost.exe (1264): S-1-5-21-4249217695-1663262354-3778214704-1110
dwm.exe (2328): S-1-5-21-4249217695-1663262354-3778214704-1110
explorer.exe (1284): S-1-5-21-4249217695-1663262354-3778214704-1110
vmtoolsd.exe (1556): S-1-5-21-4249217695-1663262354-3778214704-1110
TPAutoConnect. (2712): S-1-5-21-4249217695-1663262354-3778214704-1110
conhost.exe (2856): S-1-5-21-4249217695-1663262354-3778214704-1110
csrss.exe (2232): S-1-5-18 (Local System)
winlogon.exe (2124): S-1-5-18 (Local System)
taskhost.exe (2116): S-1-5-21-4249217695-1663262354-3778214704-1109
rdpclip.exe (2108): S-1-5-21-4249217695-1663262354-3778214704-1109
dwm.exe (2088): S-1-5-21-4249217695-1663262354-3778214704-1109
explorer.exe (2708): S-1-5-21-4249217695-1663262354-3778214704-1109
vmtoolsd.exe (1124): S-1-5-21-4249217695-1663262354-3778214704-1109
TPAutoConnect. (2012): S-1-5-21-4249217695-1663262354-3778214704-1109
conhost.exe (672): S-1-5-21-4249217695-1663262354-3778214704-1109
cmd.exe (1268): S-1-5-21-4249217695-1663262354-3778214704-1109
conhost.exe (2904): S-1-5-21-4249217695-1663262354-3778214704-1109
win64dd.exe (2836): S-1-5-21-4249217695-1663262354-3778214704-1109
```

Following the same process for analyzing DLLs in our second image, we identify 1990 instances. Again, this number can be reduced significantly by sorting and removing duplicates thus reducing the population of interest to 397.

```
$ vol.py -f win2008r2-03-s1.img dlllist | grep "^0x" | cut -c 20-37,57- | wc -l
Volatility Foundation Volatility Framework 2.3.1
1990

$ vol.py -f win2008r2-03-s1.img dlllist | grep "^0x" | cut -c 20-37,57- | sort
| uniq -i | wc -l
Volatility Foundation Volatility Framework 2.3.1
397
```

Next we create a file containing the 397 DLLs in the same format as our DLL baseline file. As a double check, the number of lines was counted using *wc*.

```
$ vol.py -f win2008r2-03-s1.img dlllist | grep "^0x" | cut -c 20-37,57- | sort
| uniq -i > dll-03.lst
Volatility Foundation Volatility Framework 2.3.1
$ cat dll-03.lst | wc -l
397
```

By applying the baseline against the DLL list, we can reduce the population of DLLs to review to 37. This is a significant improvement over reviewing 1990.

```
$ cat dll-03.lst dll-baseline-01.lst dll-baseline-01.lst | sort | uniq -iu | wc
-l
```

Gordon Fraser, Gordon.fraser@ctipc.com

```
37
```

In our analysis, we took advantage of several properties of the *uniq* command to perform the comparison. The *-i* parameter was used to ignore case. The *-u* parameter directed uniq to only print lines that are unique. To ensure the uniqueness was only due to the contents of the *dll-03.lst* file, the baseline file was added twice.

In reviewing the 37 DLLs, they look reasonable. Only one entry, *win64dd.exe*, should be removed as it was introduced by the memory capture process. Since they come from a clean image, a new baseline for DLLs can be created by merging the two lists.

```
$ cat dll-03.lst dll-baseline-01.lst | sort | uniq -i | wc -l
406
$ cat dll-03.lst dll-baseline-01.lst | sort | uniq -i >dll-baseline-02.lst
$ cat dll-baseline-02.lst | wc -l
406
```

In checking the module count for the domain attached memory image, 153 modules were found. Following the same process as with the DLLs, we quickly narrow down the modules not in the baseline to 7. This provides a much more manageable number of modules to investigate.

```
$ vol.py -f win2008r2-03-s1.img modules | cut -c 20-40,60- | grep "0x" | sort |
uniq -i | wc -l
Volatility Foundation Volatility Framework 2.3.1
153

$ vol.py -f win2008r2-03-s1.img modules | cut -c 20-40,60- | grep "0x" | sort |
uniq -i > module-03.lst
Volatility Foundation Volatility Framework 2.3.1

$ cat module-03.lst | wc -l
153
$ cat module-03.lst module-base-01.lst module-base-01.lst | sort | uniq -iu |
wc -l
7
```

The modules, which were not in our baseline, are shown below. *Win64dd.sys* is an artifact of the memory capture process and does not belong in the baseline, as shown below. The other modules do.

```
$ cat module-03.lst module-base-01.lst module-base-01.lst | sort | uniq -iu
RDPDD.dll                      0x48000 \SystemRoot\System32\RDPDD.dll
rdpdr.sys                      0x2e000 \SystemRoot\System32\drivers\rdpdr.sys
RDPWD.SYS                      0x39000 \SystemRoot\System32\Drivers\RDPWD.SYS
spsys.sys                      0x71000 \SystemRoot\system32\drivers\spsys.sys
tdtcp.sys                       0xb000 \SystemRoot\system32\drivers\tdtcp.sys
tssecsrv.sys                    0xf000
\SystemRoot\System32\DRIVERS\tssecsrv.sys
win64dd.sys                    0x11000 \??\C:\temp\windd\win64dd.sys
```

Gordon Fraser, Gordon.fraser@ctipc.com

2.4.3. Review Network Artifacts

Analysis of the network artifacts begins by running the *netscan* plugin.

```
$ vol.py -f win2008r2-03-s1.img netscan | cut -c 12-18,21-40,50-63,86-112 |
sort | uniq -w 20
Volatility Foundation Volatility Framework 2.3.1
Proto  Local Address          Foreign Addr    Pid      Owner
TCPv4  0.0.0.0:135            0.0.0.0:0       660      svchost.exe
TCPv4  0.0.0.0:3389           0.0.0.0:0       1412     svchost.exe
TCPv4  0.0.0.0:445            0.0.0.0:0       4        System
TCPv4  0.0.0.0:47001          0.0.0.0:0       4        System
TCPv4  0.0.0.0:49152          0.0.0.0:0       368      wininit.exe
TCPv4  0.0.0.0:49153          0.0.0.0:0       740      svchost.exe
TCPv4  0.0.0.0:49154          0.0.0.0:0       796      svchost.exe
TCPv4  0.0.0.0:49173          0.0.0.0:0       472      services.exe
TCPv4  0.0.0.0:49174          0.0.0.0:0       1472     svchost.exe
TCPv4  0.0.0.0:49178          0.0.0.0:0       480      lsass.exe
TCPv4  -:0                    24.217.239.1    480      lsass.exe
TCPv4  192.168.139.102:139    0.0.0.0:0       4        System
TCPv6  -:0                    18d9:ef0d:80:ffff:0 CLOSED           48
TCPv6  :::135                 :::0            660      svchost.exe
TCPv6  :::3389                :::0            1412     svchost.exe
TCPv6  :::445                 :::0            4        System
TCPv6  :::47001               :::0            4        System
TCPv6  :::49152               :::0            368      wininit.exe
TCPv6  :::49153               :::0            740      svchost.exe
TCPv6  :::49154               :::0            796      svchost.exe
TCPv6  :::49173               :::0            472      services.exe
TCPv6  :::49174               :::0            1472     svchost.exe
TCPv6  :::49178               :::0            480      lsass.exe
UDPv4  0.0.0.0:0              *:*             1472     svchost.exe
UDPv4  0.0.0.0:123            *:*             848      svchost.exe
UDPv4  0.0.0.0:4500           *:*             796      svchost.exe
UDPv4  0.0.0.0:500            *:*             796      svchost.exe
UDPv4  0.0.0.0:5355           *:*             932      svchost.exe
UDPv4  127.0.0.1:57762        *:*             480      lsass.exe
UDPv4  127.0.0.1:65282        *:*             796      svchost.exe
UDPv4  192.168.139.102:137    *:*             4        System
UDPv6  :::0                   *:*             1472     svchost.exe
UDPv6  :::123                 *:*             848      svchost.exe
UDPv6  :::4500                *:*             796      svchost.exe
UDPv6  :::500                 *:*             796      svchost.exe
UDPv6  :::5355                *:*             932      svchost.exe
```

Based on the output from *netscan* and comparing it with the baseline, the ports can be divided into three categories -- general ports, ports used when the server is not domain attached, and ports used when a server is domain attached. These are summarized in Table 6. Additional work would need to be done to validate this list with additional configurations including an IIS server, domain controller, and SQL server.

Port	Protocol	Service/Process	
General Ports			

Port	Protocol	Service/Process	
123	udp	svchost.exe (LocalService)	Windows Time Services
135	tcp	RPCSS	Terminal Services
137	udp	System	NETBIOS Name Resolution
138	tcp	System	NETBIOS Datagram Service
139	tcp	System	NETBIOS Session Service
445	tcp	System	SMB
3389	tcp		terminal services
5355	udp	svchost.exe (NetworkService)	Local Link Multicast Name Resolution (LLMNR)
47001	tcp	System	Windows Remote Management listener (WINRM)
49152	tcp	wininit.exe	
49153	tcp	svchost.exe (LocalServiceNetworkRestricted)	
49154	tcp	svchost.exe (NetworkService)	
Server Not domain Attached			
49155	tcp	lsass.exe	
49156	tcp	services.exe	
Server Domain Attached			
49173	tcp	services.exe	
49174	tcp	svchost.exe (NetworkServiceNetworkRestricted)	
49178	tcp	lsass.exe	
4500	udp	netsvcs	
500	udp	netsvcs	
57762	udp	lsass.exe	
65282	udp	netsvcs	

Table 6: Revised Baseline of open ports

Caution must be exercised when using the network ports in the baseline. When looking at ports in the dynamic port range a different dynamic port might be assigned in a different server configuration, if another process already is using the port.

3. Conclusion

This paper laid out a process for building a baseline for memory analysis. The baseline documented here is only a start. More work would have to be done to validate and build upon it using additional different Windows Server 2008 R2 configurations. A

similar process could be used to build baselines for Windows 7, Windows 8, and Windows Server 2012.

The value of a baseline was demonstrated during the process of building and testing one. The baseline, by documenting the expected processes, allowed us to quickly identify processes that were not in the baseline but should have been. We could just as easily have used the baseline against the memory image of a compromised system to identify the unexpected. In looking at the processes in the process and DLL lists, we were able to quickly focus on a few processes out of dozens. The value of using a baseline was even more dramatic when we reduced the list of DLLs to examine from 1990 to 37 and the list of kernel modules from 153 to 7.

A generic baseline can be helpful for general analysis of memory images from multiple sources. For an organization, a baseline tailored to their standard configurations could save considerable time in analysis. In our baseline, we saw the effects of the baseline on artifacts introduced by our implementing the servers under VMware workstation. Other artifacts could be expected for an organization's standard configurations like anti-virus software. Thus, establishing a baseline could save the analyst considerable time.

Gordon Fraser, Gordon.fraser@ctipc.com

4. References

IANA. (2014). Service Name and Transport Protocol Port Number Registry. Retrieved August 3, 2014 from http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

Lee, Rob & Tilbury, Chad. (2013). Incident Response and Memory Analysis. The SANS Institute.

Ligh, Michael Hale, Case, Andrew, Levy, Andrew, and Walters, Aaron. (2014). The Art of Memory Forensics. Indianapolis, ID: Wiley.

Microsoft Developer Network. (2014). Obtaining Data from the Local Computer. Retrieved August 3, 2014 from http://msdn.microsoft.com/en-us/library/aa384424(v=vs.85).aspx.

Microsoft Knowledge Base (2013a). Well-known security identifiers in Windows Operating Systems, Article ID: 243330. Retrieved July 27, 2014 from http://support.microsoft.com/kb/243330.

Microsoft Knowledge Base (2013b). Service overview and network port requirements for Windows, Article ID: 832017. Retrieved July 27, 2014 from http://support.microsoft.com/kb/832017.

Microsoft Knowledge Base (2013c). The default dynamic port range for TCP/IP has changed in Windows Vista and in Windows Server 2008, Article ID: 929851. Retrieved July 27, 2014 from http://support.microsoft.com/kb/929851.

Olsen, Patrick. (2014). Know your Windows Processes or Die Trying. Retrieved July 30, 2014 from http://sysforensics.org/2014/01/know-your-windows-processes.html.

Pilkington, Mike & Lee, Rob. (2014). SANS DFIR Digital Forensic & Incident Response Poster, Spring 2014, 29th Edition. The SANS Institute.

Russinovich, Mark, Solomon, David A., and Ionsecu, Alex. (2012a). Windows Internals. 6th Edition, Part 1. Redmond, WA: Microsoft Press.

Russinovich, Mark, Solomon, David A., and Ionsecu, Alex. (2012b). Windows Internals. 6th Edition, Part 2. Redmond, WA: Microsoft Press.

Gordon Fraser, Gordon.fraser@ctipc.com

The Cable Guy. (2006, November). Link-Local Multicast Name Resolution. Retrieved August 3, 2014 from http://technet.microsoft.com/library/bb878128.

VMware. (2011) vSphere Virtual Machine Administration Guide. Palo Alto, CA: VMware, Inc.

VMware Knowledge Base. (2014, June 24). Processes started by the View Agent and View Client (1015677). Retrieved July 27, 2104 from http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1015677.

Startup Programs Database. (2014a). TrustedInstaller.exe Information. Retrieved August 2, 2014 from http://www.bleepingcomputer.com/startups/TrustedInstaller.exe-25809.html.

Startup Programs Database. (2014b). Sppsvc.exe Information. Retrieved August 2, 2014 from http://www.bleepingcomputer.com/startups/sppsvc.exe-25807.html.

Startup Programs Database. (2014c). Dllhost.exe Information. Retrieved August 2, 2014 from http://www.bleepingcomputer.com/startups/dllhost.exe-25641.html.

Startup Programs Database. (2014d). Msdtc.exe Information. Retrieved August 2, 2014 from http://www.bleepingcomputer.com/startups/msdtc.exe-3339.html.

Startup Programs Database. (2014e). spoolsv.exe Information. Retrieved August 2, 2014 from http://www.bleepingcomputer.com/startups/spoolsv.exe-25571.html.

Volatility Wiki. (2013). Volatility 2.3: Volatility Basic Usage. Retrieved August 9, 2014 from http://code.google.com/p/volatility/wiki/VolatilityUsage23.

Gordon Fraser, Gordon.fraser@ctipc.com