Table of Contents

Registry Analysis - Questions

This lab simulates the common scenario of needing to hunt for malware on a system without Indicators of Compromise (IOCs) or other clues related to specific features of malware that may be present. In these situations, the analyst must utilize techniques that allow for quickly triaging high-value artifacts to efficiently uncover signs of an infection.

For this lab, you have been given four sets of registry hives, in the folders *1*, *2*, *3*, and *baseline*. Each set contains the SOFTWARE and SYSTEM hive from the same system at unique points in time. The hives in the numbered folders contain artifacts that reveal the presence of malware (one infection per set) whereas the baseline folder contains the hives from the system in its original state.

Use techniques that we discussed, such as batch artifact analysis with RegRipper, *diff*-based analysis, and analysis profiles, to find the malware artifact that reveals that malware present in each numbered folder set. Remember that custom analysis profiles can be made by newline-separating plugin names and then specifying the path to the profile with the -f option.

For clarity: there is one infection you must find the artifacts of **per set**, not per hive.

Bonus 1: Updating a Plugin Template

To complete this part, you will need to analyze the "virus-SOFTWARE" hive in the *PluginWriting* folder with a custom RegRipper plugin that you will write.

The "virus" writes to the key "Microsoft\\DirectX\\virusdomain" and its "lastdomain" value holds the URL of the last domain the virus contacted.

You must write a RegRipper plugin that can recover the last domain contacted.

Suggestions and Hints:

To avoid creating a plugin from scratch, use the *wab* plugin as a starting point and modify it to retrieve the key and value related to this lab. This file is stored at */home/training/regripper/plugins* in your provided virtual machine. Start by making a copy of this file under the same folder, but with a different name, such as virusdomain.pl. To register your new plugin, you must change the line of your copied file that says “package wab;” to be “package virusdomain;” - assuming you named your file virusdomain.pl as suggested. You would then run your plugin by passing *-p virusdomain* to RegRipper.

Bonus 2: Plugin Writing (**Warning**: Spoilers for the main scenario in this description)

After finding a malicious browser helper object in the registry sets, you have been tasked with automatically finding its malicious DLL in other registry hives across your corporate environment. To accomplish this task, update the *bho* plugin to automatically alert if the malicious DLL found in Part 1 is present.

Registry Analysis – Answers

There were several approaches to solving this lab, all of which had differing levels of automation.

Approach 1: No Automation

This lab could have been solved with a manual approach by running the malware detection plugins discussed in the slides. If this approach was taken, you would have had to either manually:

1) Determine which output lines showed malicious executables
2) Compare a particular plugin's output against an infected hive with that of the output against the baseline hive

Either process would have found the malicious code. The following shows the output of the *appcertdlls* plugins against the baseline SYSTEM hive as well as the SYSTEM hive from hive set 1.

```
# rip.pl -r 1/1.system -p appcertdlls
Launching appcertdlls v.20200427
ControlSet001\Control\Session Manager\AppCertDlls
LastWrite Time: 2017-11-22 22:46:58Z
fltMsfc - C:\WINDOWS\system32\cidaec32.dll

# rip.pl -r baseline/system -p appcertdlls
Launching appcertdlls v.20200427
ControlSet001\Control\Session Manager\AppCertDlls not found.
```

As shown, the baseline has no AppCertDlls registered, but set 1 does.

Approach 2: Partial Automation

Beyond manual inspection of plugin output against the clean and infected hives, the *diff* command can be used to automatically display any changes. The following shows this analysis for the *svc* plugin against the SYSTEM hive from the baseline set and set 2.

```
#  rip.pl -r baseline/system -p svc > svc.before
Launching svc v.20200525
# rip.pl -r 2/2.system -p svc > svc.after
Launching svc v.20200525
# diff svc.before svc.after
5c5,6
< 2017-11-22 22:16:12Z,Dhcp\Parameters,,%SystemRoot%\System32\dhcpcsvc.dll,,,,
---
> 2017-11-22 22:44:14Z,lanmandrv,lanmandrv,\??\C:\WINDOWS\System32\lanmandrv.sys,Kernel driver,System Start,,IPSEC driver
> 2017-11-22 22:34:43Z,Dhcp\Parameters,,%SystemRoot%\System32\dhcpcsvc.dll,,,,
```

In this output, we can see that a parameter to the DHCP service has changed, and that a new service, *lanmandrv*, has been installed. The change to DHCP looks benign, due to DHCP parameters often changing, but to be thorough in a real investigation, you would want to verify that dhcpcsvc.dll has not been modified on disk.

On the other hand, the new *lanmandrv* service loads itself as a kernel driver and has registered as a suspicious name related to IPSEC. Searching online for *lanmandry.sys*, the name of the kernel driver, leads to many hits related to a powerful rootkit *Laqma*:

https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj~Laqma-A/detailed-analysis.aspx

Approach 3: Using Profiles

Using profiles as discussed during the lecture would have allowed for gathering many malware artifact types with only one RegRipper invocation per hive. In the following example, we used a profile with persistence plugins related to SOFTWARE and SYSTEM hives included. This allowed us to quickly gather these artifacts into one file per-hive.

```
# rip.pl -r baseline/system -f persistence >> base.per
# rip.pl -r baseline/software -f persistence >> base.per
# rip.pl -r 3/3.system -f persistence >> 3.per
# rip.pl -r 3/3.software -f persistence >> 3.per
```

We then ran the diff command on them (note: for this to work, you must analyze hives in the same order for each system):

```
# diff base.per 3.per
5c5
< 2017-11-22 22:16:12Z,Dhcp\Parameters,,%SystemRoot%\System32\dhcpcsvc.dll,,,,
---
> 2017-11-22 22:28:30Z,Dhcp\Parameters,,%SystemRoot%\System32\dhcpcsvc.dll,,,,
382c382,389
< Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects not found.
---
> Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
> LastWrite Time Wed Nov 22 22:30:47 2017 (UTC)
>
> {0000AC13-3487-1583-C4BE-BE6A839DB000}
>   Class     => Microsoft Shared Library Object Version
>   Module    => C:\WINDOWS\system32\mfc42dx1.dll
>   LastWrite => Wed Nov 22 22:30:47 2017
>
405d411
<   VMware User Process - "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
406a413
>   VMware User Process - "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
```

Examining the *diff* output shows 3 sets of artifacts. The first is for the DHCP time changes we already discussed and the last is for VMWare. Examining the VMware lines shows that the values are the same show, and inspection of the *.per files show that this output only happened as they were on different lines. We can safely ignore this. The remaining lines appear to show that a BHO is active in set 3 that is not present in the baseline. Running the bho plugin against 3.software confirms this:

```
Launching bho v.20130408
bho v.20130408
(Software) Gets Browser Helper Objects from Software hive

Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
LastWrite Time Wed Nov 22 22:30:47 2017 (UTC)

{0000AC13-3487-1583-C4BE-BE6A839DB000}
  Class     => Microsoft Shared Library Object Version
  Module    => C:\WINDOWS\system32\mfc42dx1.dll
  LastWrite => Wed Nov 22 22:30:47 2017
```

Notes and Thoughts

As shown, there are a number of ways to approach analysis of registry hives looking for quick wins during triage. Particularly when other hives, such as from local VSS stores or other systems in the same environment, can be used to drive diff-based analysis, most of the task can be automated.

In situations where diff-based analysis is not possible, the use of RegRipper profiles to quickly gather key artifacts against infected hives can often lead to quick identification of the malware present. Combined with bash scripting, the use of profiles allows for gathering all the artifacts from all hives gathered from a system in one command invocation. The following is a basic example of this against the '3' hive set:

```
$ for hive in $(ls 3/*); do perl rip.pl -r $hive -f persistence >> 3.per; done
```

This for loop invokes RegRipper on each hive in the folder and saves the output to the 3.per file.

Finally, when combined with the use of timelines, which we will discuss shortly, this technique becomes even more effective.

Bonus 1: Updating a Plugin Template

To complete this bonus, you simply needed to update the key path that the plugin analyzed and updated one of the value printing lines with the virus' value. The following is a diff of the existing *wab* plugin that shows the changes needed:

```
# diff plugins/wab.pl plugins/virusdomain.pl
16c16
< package wab;
---
> package virusdomain;
47c47
<     my $key_path = "Microsoft\\WAB\\DLLPath";
---
>     my $key_path = "Microsoft\\DirectX\\virusdomain";
56,57c56,57
<                         $def = $key->get_value("")->get_data();
<                         ::rptMsg("(Default) value = ".$def);
---
>                         $def = $key->get_value("lastdomain")->get_data();
>                         ::rptMsg("last domain = ".$def);
```

With these changes, the plugin then produces the following output:

```
# rip.pl -p virusdomain -r PluginWriting/virus-SOFTWARE
<snip>
last domain = www.blackhat.cn
```

Note: You did not have to change the line with rptMsg to say “last domain”, it just makes the output nicer to read.

Bonus 2: Plugin Writing

There is no one correct answer to this bonus since it involves adding logic to existing plugins. As an example, the following shows logic that could have been added to the *bho* plugin in order to specifically alert about the malicious DLL’s presence:

```
diff --git a/plugins/bho.pl b/plugins/bho.pl
index b274226..894723f 100644
--- a/plugins/bho.pl
+++ b/plugins/bho.pl
@@ -92,6 +92,9 @@ sub pluginmain {
                                        }

                                        foreach my $b (keys %bhos) {
+                                               if ($bhos{$b}{module} == "C:\WINDOWS\system32\mfc42dx1.dll") {
+                                               ::rptMsg("MALICIOUS BHO ASSOCIATED WITH SILENT BANKER FOUND!");
+                                               }
                                                ::rptMsg($b);
                                                ::rptMsg("  Class     => ".$bhos{$b}{class});
                                                ::rptMsg("  Module    => ".$bhos{$b}{module});
@@ -114,4 +117,4 @@ sub pluginmain {
                }
        }
 }
-1;
\ No newline at end of file
+1;
```

Event Log Analysis - Questions

Scenario

Joe Forensics was assigned on a case for a small company named Manager Mart who was attacked by an APT actor called the Kitty Lovers Group (KLG). This group was stealing information from Manager Mart, including personal pictures of their kitty family members and posting it on the web. The group was within the Manager Mart infrastructure for a month's time (from what the client knew) before Joe took the case.

While waiting for a memory sample that was taken from the Exchange server of Manager Mart, Joe was told that the IT admin would send him a VPN installer to gain early access. What Joe didn't know was that the KLG knew they were being investigated and were impersonating Manager Mart employees. KLG used this impersonation capability in order to send Joe a RAT that would destroy his evidence and investigation.

After installing the received "cisco_vpn.exe" program, Joe's computer began acting strange. Joe was suspicious, since the executable disappeared after double-clicking it, but figured it might just be part of the installation process. He continued working on the event logs and registry files that were taken from another machine from Manager Mart prior. However, before he could get very far, files began disappearing, FTK crashed and then a message box from the attacker appeared stating that his evidence had been destroyed.

Select files were taken from Joe's machine and given to you for analysis.

Given Evidence

Disk.7z – A compressed copy of select files from Joe's system

evtexport_output.7z – A compressed archive of the processed Application, Security, and System event logs

mft.body and mft.csv – A body file and the corresponding comma-separated timeline of the disk's MFT

Part 1

1. Which well-known RAT is the cisco_vpn.exe program?
   a. What is the date and time that it appeared on Joe's machine?
   b. When did it run?
2. What is the r.exe program that was uploaded from the attacker onto Joe Forensics' machine?
   a. When was the date and time that it appeared on Joe's machine?
3. Is security logging enabled on this machine? If so, which events can you expect to find?
4. Use the processed Security events to answer the following:

a. Is there any evidence of the attacker's backdoor?
b. Which running process(es) would you investigate if you had a memory sample of this system?

Part 2

Inside of the Evidence files in the Recycle Bin folder of Joe's account (hint: start looking under *C/RECYCLER/S-1-5-21-1177238915-1336601894-1801674531-500/*) is a copy of the Windows Defender operational event log file. The name of this file is *Microsoft-Windows-Windows Defender%4Operational.evtx*. Run *evtxdump.pl*, which is in your $PATH, so that you can run it from any directory, against this event log file to answer the following questions:

1. Which Trojan was the attacker using on this machine?
2. Where was the Trojan builder located on the machine?

Part 3

Also inside the Recycle Bin is a Security Event log (*Security.evtx*) that Joe was investigating before his files were deleted. Analyze this file with evtxdump.pl to answer the following questions:

1. Is there any evidence of a victim's machine (IP address 172.16.1.135) connecting to the machine Joe was investigating?
2. When was the last time that machine was connected?

Event Log Analysis – Answers

Part 1

1. Which well-known RAT is the cisco_vpn.exe program?

In order to answer this question, you must recover the cisco_vpn executable. Searching for cisco_vpn.exe in the provided *mft.csv* file reveals its location as *Disk/JoeForensics/C/Documents and Settings/Administrator/Desktop/cisco_vpn.exe* and it has a MD5 hash of *086ce6675cfb293d5947776d8638bd60*. Searching this hash on VirusTotal (https://www.virustotal.com/#/file/614f8ea3f7333c3da6a7c3c0733ca497021c98d6999eaf6580b39f9f4791ab6d/detection) reveals it to be the DarkKomet RAT.

a. What is the date and time that it appeared on Joe's machine?

Examination of the MFT timeline shows a file named cisco_vpn.7z being created in the User's Downloads folder:

```
"/Documents and Settings/Administrator/My Documents/Downloads/cisco_vpn.7z","2014-09-11 18:58:42.885242"
```

Roughly seven minutes later, the cisco_vpn.exe file appears in the user's Desktop folder:

```
"/Documents and Settings/Administrator/Desktop/cisco_vpn.exe","2014-09-11 19:05:44.782711"
```

b. When did it run?

According to the output of RegRipper's *userassist* plugin against the user's NTUSER.dat hive, cisco_vpn.exe ran once on September 11th:

```
Thu Sep 11 19:19:35 2014 Z
  UEME_RUNPATH:C:\Documents and Settings\Administrator\Desktop\cisco_vpn.exe (1)
```

2. What is the r.exe program that was uploaded from the attacker onto Joe Forensics' machine?

We can answer this question through strings analysis or by checking the hash of the file on VirusTotal (https://www.virustotal.com/en/file/43ed8acbb898e4d6a53862e88b18f3b6eac82c4240f3afe8babf3e8e1300266e/analysis/). This analysis would have led us to determine that the tool is WinRAR.

a. When was the date and time that it appeared on Joe's machine?

Searching this file across the timeline reveals:

```
/WINDOWS/system32/1076/r.exe","2014-09-11 19:37:48.116478"
```

3. Is security logging enabled on this machine?

This system is configured to log all possible events:

```
# rip.pl -p auditpol -r <path/to/extracted/Disk.7z>/JoeForensics/C/WINDOWS/system32/config/SECURITY
<snip>
(Security) Get audit policy from the Security hive file

auditpol
Policy\PolAdtEv
LastWrite Time Wed Sep 10 16:57:35 2014 (UTC)

Auditing is enabled.
        Audit System Events          = S/F
        Audit Logon Events           = S/F
        Audit Object Access          = S/F
        Audit Privilege Use          = S/F
        Audit Process Tracking       = S/F
        Audit Policy Change          = S/F
        Audit Account Management     = S/F
        Audit Dir Service Access     = S/F
        Audit Account Logon Events   = S/F
```

4. Look at the events in the SecEvent.Evt log and the exported output from evtexport (SecEvent_exported.txt)

a. Is there any evidence of the attacker's backdoor?

Yes. We see events for the cisco_vpn.exe process that include its start execution time as well as the fact that it has SeTakeOwnershipPrivilege privileges.

```
Event number            : 553
Creation time           : Sep 11, 2014 19:19:35 UTC
Written time            : Sep 11, 2014 19:19:35 UTC
Event type              : Success Audit event (8)
User security identifier  : S-1-5-21-1177238915-1336601894-1801674531-500
Computer name           : USER-6713BE3AFE
Source name             : Security
Category message filename   : %SystemRoot%\System32\MsAuditE.dll
Event category          : Detailed Tracking (5)
Event identifier        : 0x00000250 (592)
Message         filename                                                        :
%SystemRoot%\System32\MsAuditE.dll;%SystemRoot%\System32\xpsp2res.dll;%SystemRoot%\System32\xpsp3
res.dll
Number of strings       : 6
String: 1               : 2408
String: 2               : C:\Documents and Settings\Administrator\Desktop\cisco_vpn.exe
String: 3               : 1740
String: 4               : Administrator
String: 5               : USER-6713BE3AFE
Message string          : A new process has been created:
    New Process ID: 2408
    Image File Name:    C:\Documents and Settings\Administrator\Desktop\cisco_vpn.exe
    Creator Process ID: 1740
    User Name:  Administrator
```

```
    Domain:     USER-6713BE3AFE
    Logon ID:       (0x0,0x27162)

Event number            : 554
Creation time           : Sep 11, 2014 19:19:35 UTC
Written time            : Sep 11, 2014 19:19:35 UTC
Event type          : Success Audit event (8)
User security identifier    : S-1-5-21-1177238915-1336601894-1801674531-500
Computer name           : USER-6713BE3AFE
Source name         : Security
Category message filename   : %SystemRoot%\System32\MsAuditE.dll
Event category          : Privilege Use (4)
Event identifier        : 0x00000242 (578)
Message         filename                                                    :
%SystemRoot%\System32\MsAuditE.dll;%SystemRoot%\System32\xpsp2res.dll;%SystemRoot%\System32\xpsp3
res.dll
<snip>
Message string          : Privileged object operation:
    Object Server:  Security
    Object Handle:  232
    Process ID: 2408
    Primary User Name:  Administrator
    Primary Domain: USER-6713BE3AFE
    Primary Logon ID:   (0x0,0x27162)
    Client User Name:   -
    Client Domain:  -
    Client Logon ID:    -
    Privileges: SeTakeOwnershipPrivilege
```

You will also start to notice some other processes, such as notepad with process ID (PID) 1764, that have this unexpected privilege and another process called C:\Documents and Settings\All Users\Start Menu\Programs\MSDCSC\msdcsc.exe with PID 324:

b. Which running process(es) would you investigate if you had memory sample?

Process IDs 2408, 324, 1764

Part 2

Inside of the Recycle Bin is a copy of the Windows Defender operational event log file. The name of this file is *Microsoft-Windows-Windows Defender%4Operational.evtx*. Run evtxdump.pl, which is in your $PATH so that you can run it from anywhere, against this event log file to answer the following question:

1. Looking at the Windows Defender event logs:
   a. What Trojan was the attacker using on this machine?

      TrojanDownloader:Win32/Small.AJI

   b. Where was the Trojan builder located on the machine?

      C:\Users\IEUser\Desktop\tempsample\DarkComet.exe

```
# evtxdump.pl Microsoft-Windows-Windows\ Defender%4Operational.evtx |less -I

<TimeCreated SystemTime="2014-09-11T14:46:07.4540Z" />
<EventRecordID>3</EventRecordID>
<Correlation />
<Execution ProcessID="572" ThreadID="816" />
<Channel>Microsoft-Windows-Windows Defender/Operational</Channel>
<Computer>IE9Win7</Computer>
<Security UserID="S-1-5-18" /></System>
<EventData>
<Data Name="Product Name">%%827</Data>
<Data Name="Product Version">6.1.7600.16385</Data>
<Data Name="Detection ID">{6964FD97-80C3-4F1D-AB7D-312006EF9C68}</Data>
<Data Name="Detection Source Index">3</Data>
<Data Name="Detection Source">%%818</Data>
<Data Name="Unused"></Data>
<Data Name="Process Name"></Data>
<Data Name="Domain"></Data>
<Data Name="User"></Data>
<Data Name="SID"></Data>
<Data Name="Threat Name">TrojanDownloader:Win32/Small.AJI</Data>
<Data Name="Threat ID">172769</Data>
<Data Name="Severity ID">5</Data>
<Data Name="Category ID">4</Data>
<Data
Name="FWLink">http://go.microsoft.com/fwlink/?linkid=37020&amp;name=TrojanDownloader:Win3
2/Small.AJI&amp;threatid=172769</Data>
<Data                                                                    Name="Path
Found">containerfile:C:\Users\IEUser\Desktop\tempsample\DarkComet.exe;file:C:\Users\IEUse
r\Desktop\tempsample\DarkComet.exe-&gt;[RSRCEmb];process:pid:1292</Data>
```

Part 3

1. Is there any evidence of a victim's machine (IP address 172.16.1.135) connecting to the machine Joe was investigating?

```
<TimeCreated SystemTime="2014-09-10T16:23:52.276Z" />
<EventRecordID>2939</EventRecordID>
<Correlation />
<Execution ProcessID="472" ThreadID="1912" />
<Channel>Security</Channel>
<Computer>IE9Win7</Computer>
<Security /></System>
<EventData>
<Data Name="SubjectUserSid">S-1-0-0</Data>
<Data Name="SubjectLogonId">0x0000000000000000</Data>
<Data Name="TargetUserSid">S-1-5-7</Data>
<Data Name="TargetUserName">ANONYMOUS LOGON</Data>
<Data Name="TargetDomainName">NT AUTHORITY</Data>
<Data Name="TargetLogonId">0x0000000000164026</Data>
<Data Name="LogonType">3</Data>
<Data Name="LogonProcessName">NtLmSsp </Data>
<Data Name="AuthenticationPackageName">NTLM</Data>
<Data Name="WorkstationName">USER-6713BE3AFE</Data>
<Data Name="TransmittedServices">-</Data>
<Data Name="LmPackageName">NTLM V1</Data>
<Data Name="KeyLength">0</Data>
<Data Name="ProcessId">0x00000000</Data>
<Data Name="ProcessName">-</Data>
<Data Name="IpAddress">172.16.1.135</Data>
```

2. When was the last time that machine was connected?

   The last occurrence appears to be at <TimeCreated SystemTime="2014-09-11T15:21:41.8763Z" />

Finding a Malware Infection - Questions

Scenario

Your friend recently had his computer crash and was forced to reinstall his operating system in order to make the machine usable again. Luckily, your friend regularly backs up his games and game data, important document files, and other personal data to a series of USB drives.
After reinstalling Windows, he copied his backup files into the new install and then reinstalled his games. Unfortunately, soon after this, he noticed that his system was acting strange. In particular, a "black box" would often pop up and then quickly disappear.

Intrigued, you decide to help your friend and determine the root cause of this issue. To start, you generate a disk image of your friend's machine as well as a timeline filtered to events from the current year (2017).

Note: While you are strongly encouraged to use the timeline file to get through the lab efficiently, you may wish examine files directly from the hard drive image. To assist in this task, we have placed a bash script that will mount the disk image for you in a read-only manner. To activate the mountpoint, simply execute the *mount_lab_3.sh* script under */home/training*. You will need to run this script as root or through sudo. Upon successful completion, the disk image will be mounted and accessible under the */mnt/lab_3/* folder.

Questions

1) How many USB drives did your friend use while restoring files to his system?

2) What are the major categories of files that your friend restored to his computer?

3) What is the persistence mechanism used by the installed malware?

4) What is the malicious application that has been installed on the system?

5) Which script file initially infected the machine after being ran? Hint: It is not the same as the malware that runs in a persistent manner.

6) Assume you did not recover the infection script in Question 5. Is it possible to find the original name of the executable component of the malware when it was first copied from the USB drive?

Finding a Malware Infection - Answers

Notes: (Spoilers!)

Due to the hints and questions given, this lab could have been started with direct analysis of the artifacts (such as the USB devices in question 1). This would have given a good timeframe to pivot analysis inside the timeline.

Otherwise, pivoting to look for execution of suspicious and/or out-of-place executables would have helped narrow down to the malware activity.

For example, analysis of the *userassist* portion of the registry would have quickly led to the discovery of oddly named executables being run. These were related to the games folder. Similarly, the *schtasks* binary ran very shortly (30 seconds) before *launch.exe* executed for the first time. Pivoting to this point in the timeline would have immediately answered many of the questions surrounding the malware, such as the malware being *launch.exe*, the reference to *debug.log*, and so on.

Timelines are one of the most powerful tools in DFIR, but understanding how events correlate and how to pivot smartly is the only way to make effective use of them.

1) How many USB drives did your friend use while restoring files to his system?

Through analysis of the timeline or through RegRipper analysis of the SYSTEM hive from the disk image, you would find evidence of three thumb drives:

```
# rip.pl -r system -p usbstor
<snip>
Disk&Ven_SanDisk&Prod_Cruzer&Rev_1.00 [Wed Nov 29 22:37:58 2017]
  S/N: 4C530001091022100104&0 [Wed Nov 29 22:38:02 2017]
  Device Parameters LastWrite: [Wed Nov 29 22:38:02 2017]
  LogConf LastWrite          : [Wed Nov 29 22:37:58 2017]
    FriendlyName    : SanDisk Cruzer USB Device
    ParentIdPrefix: 8&25f26d71&0

Disk&Ven_SanDisk&Prod_Cruzer&Rev_1.27 [Wed Nov 29 23:28:45 2017]
  S/N: 4C530001051023109270&0 [Wed Nov 29 22:01:55 2017]
  Device Parameters LastWrite: [Wed Nov 29 22:01:55 2017]
  LogConf LastWrite          : [Wed Nov 29 22:01:49 2017]
    FriendlyName    : SanDisk Cruzer USB Device
    ParentIdPrefix: 8&3b096db3&0
  S/N: 4C530001111023109282&0 [Wed Nov 29 23:28:49 2017]
  Device Parameters LastWrite: [Wed Nov 29 23:28:49 2017]
  LogConf LastWrite          : [Wed Nov 29 23:28:45 2017]
    FriendlyName    : SanDisk Cruzer USB Device
    ParentIdPrefix: 8&be54833&0
```

2) What are the major categories of files that your friend restored to his computer?

Analysis of the timeline would reveal many LNK files associated with RTF and PDF files being opened on the infected system. There would also be many files and executables associated with a "gamesfiles" folder on your friend's Desktop.

3) What is the persistence mechanism used by the installed malware?

Analysis of the timeline would have revealed a Scheduled Task being created along with several associated events within one second of each other (note that the fields have been filtered for easier reading):

```
11/29/2017,23:27:23,MACB,UNKNOWN Content Modification
Time,/WINDOWS/Prefetch/SCHTASKS.EXE-0CBF6A11.pf
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/Documents and
Settings/Administrator/Application Data
11/29/2017,23:28:00,...B,UNKNOWN Creation Time,/debug.log
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/WINDOWS/system32/hid.dll
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/WINDOWS/system32/cryptdll.dll
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/WINDOWS/system32/winscard.dll
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/WINDOWS/system32/rsaenh.dll
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/WINDOWS/system32/lsasrv.dll
11/29/2017,23:28:00,.A..,UNKNOWN Last Access Time,/WINDOWS/system32/samsrv.dll
11/29/2017,23:28:00,...B,UNKNOWN Creation Time,/WINDOWS/Prefetch/LAUNCH.EXE-019DDF8E.pf
```

As can be seen, immediately after schtasks.exe executes, several events of interest happen in the next second, including:

- Creation of a file named debug.log in the root of the file system
- Many encryption related DLLs being accessed
- A Prefetch file for an application named "launch.exe"
- The "Application Data" folder being accessed

This leads to many investigate steps, including:

- Extracting debug.log, which contains valid mimikatz output
- Investigating the "Application Data" folder of the Administrator user account. Examination of this folder's contents reveals an executable named launch.exe, which matches the Prefetch file.

To determine if the scheduled task is still active, you could either/both search the timeline output for scheduled tasks and/or check the directory in the disk image for .job files.

The timeline reveals the following scheduled task, "EA Games Installer", was set to execute immediately before the previously found activity:

11/29/2017,23:27:00,UTC,....,JOB,Windows Scheduled Task Job,Scheduled to start,Administrator,USER-6713BE3AFE,Application: cmd /c "C:\Documents and Settings\Administrator\Application Data...,Application: **cmd /c "C:\Documents and Settings\Administrator\Application Data\launch.exe" privilege::debug sekurlsa::logonpasswords exit >> C:\debug.log** Scheduled by: Administrator Working directory: cmd Trigger type: DAILY,2,TSK:/**WINDOWS/Tasks/EA Games Installer.job**,-,-,winjob,comment: ; sha256_hash: 7a63f2b52097dddd51dcd1a47853cff2d83e517d81e8d66f640c8219ed53c623

4) What is the malicious application that has been installed on the system?

The md5 hash of launch.exe is a6ef79acdcd0d2eaf54d2fa39e421c05, which VirusTotal shows is a version of Mimikatz.

5) Which script file initially infected the machine after being ran? Hint: It is not the same as the malware that runs in a persistent manner.

Answering this question through the timeline likely required a careful eye. The file used to launch the infection was transferred in the game files folder and named *launch.bat*. It made a copy of another file in the game files folder called warcraft-nocd.exe. This copy was placed in the Application Data folder as launch.exe, which we know is the persistently activated mimikatz.

6) What was the original name of the executable component of the malware when it was first copied from the USB drive?

Assuming we did not find launch.bat to tell us information about warcraft-nocd.exe, we would need another indicator of its association with the infection. Since the file did not launch directly, it left no artifacts related to program execution. Similarly, its timestamps are muddled to the point that the file is not directly associated with the infection. The strongest association with warcraft-nocd.exe and launch.exe is found by examining the compile time and sha256 hashes from the timeline:

08/13/2017,15:27:48,UTC,...B,PE,PE Compilation time,Creation Time,-,USER-6713BE3AFE,pe_type,PE Type: Executable (EXE) Import hash: 214cccffb2136a0559ff0c2324b226ba,2,TSK:**/Documents and Settings/Administrator/Desktop/gamefiles/warcraft-nocd.exe**,-,-,pe,section_names: [u'.text\x00\x00\x00' u'.rdata\x00\x00' u'.data\x00\x00\x00' u'.rsrc\x00\x00\x00' u'.reloc\x00\x00']; sha256_hash: **9369b34df04a2795de083401dda4201a2da2784d1384a6ada2d773b3a81f8dad**

08/13/2017,15:27:48,UTC,...B,PE,PE Compilation time,Creation Time,-,USER-6713BE3AFE,pe_type,PE Type: Executable (EXE) Import hash: 214cccffb2136a0559ff0c2324b226ba,2,TSK:**/Documents and Settings/Administrator/Application Data/launch.exe**,-,-,pe,section_names: [u'.text\x00\x00\x00' u'.rdata\x00\x00' u'.data\x00\x00\x00' u'.rsrc\x00\x00\x00' u'.reloc\x00\x00']; sha256_hash: **9369b34df04a2795de083401dda4201a2da2784d1384a6ada2d773b3a81f8dad**

In this output, we can see that the files match exactly and share a sha_256 hash. In the full timeline, the lines are one right after the other and can be easily seen if navigating entries around *launch.exe* references.

Memory - Vanilla Sample - Questions

Profile: Win7SP1x64

This is a straightforward lab to help ensure that all students are comfortable using Volatility. You have been given a memory sample (hands-on-plugins.raw) inside of a ZIP file (hands-on-plugins.zip). Once extracted, then please use Volatility to answer the following questions.

Questions:

1. What is the process ID of the Dumpit.exe process? (hint: pslist)
2. What directory did Dumpit capture memory to? (hint: handles)
3. What was the full path of the file that Dumpit wrote? (hint: filescan and consoles)
4. What was the virtual offset of the usbccgp.sys driver? (hint: modules)
5. Which DLL files were loaded into the DumpIt process? (hint: ldrmodules)
6. What file is mapped from the VAD starting at 0xfffffa80054a6b50 in the SpotifyWebHelp process? (hint: vadinfo)
7. What port is lsass listening on? (hint: netscan)

Notes:

As an example, to run the 'pslist' plugin, the following command can be used:

```
$ volatility -f hands-on-plugins.raw --profile=Win7SP1x64 pslist
```

You just need to ensure that the *–f* option contains the real path to the memory sample.

Memory - Vanilla Sample - Answers

1. **What is the process ID of the Dumpit.exe process?**
   We can use pslist and grep for the Dumpit process:

```
$ volatility -f hands-on-plugins.raw --profile=Win7SP1x64 pslist |grep -i dump
Volatile Systems Volatility Framework 2.3_beta
0xfffffa8006843b30 DumpIt.exe        1836  1624      2       45      1       1 2012-07-19 06:50:07 UTC+0000
```

2. **What directory did Dumpit capture memory to?**
   We can use the handles plugin to see what files the dumpit process has open. Since it is writing a directory, it must have a handle to it open:

```
$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 handles -t File -p 1836
Volatile Systems Volatility Framework 2.3_beta
Offset(V)          Pid            Handle         Access Type     Details
------------------ ------ ------------------ ------------------ ---------------- -------
0xfffffa8003d41d40   1836             0x10         0x100020 File           \Device\HarddiskVolume1\Windows
0xfffffa8006871710   1836             0x1c         0x100020 File
\Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt
0xfffffa800681b070   1836             0xb4         0x1f01ff File           \Device\DumpIt
```

3. **What was the full path of the file that Dumpit wrote?**
   We can use the filescan plugin to find the memory sample path. Since we know the directory location from the previous question, we can use part of it in our grep statement:

```
$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 filescan |grep -i dumpit
Volatile Systems Volatility Framework 2.3_beta
0x0000000001401370      5      0 R--r-- \Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt\DumpIt.exe
0x0000000006cbb710      1      1 R--rw- \Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt
0x00000000082be1e0      2      1 R--rwd \Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt
0x00000000088924f0      2      1 R--rwd \Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt
0x000000000a040a80     10      0 R--r-d \Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt\DumpIt.exe
0x000000000ac5f730      1      1 RW-rw- \Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt\WIN-
TK0PBVQLQNB-20120719-065010.raw
```

   The memory sample itself doesn't show up in the dumpit.exe's handle list, because the kernel is writing it out, so we can find it in the System's handle list instead:

```
$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 handles -t File -p 4|grep -i dump
Volatile Systems Volatility Framework 2.3_beta
0xfffffa8006423730      4             0xa0c         0x12019f File
\Device\HarddiskVolume1\Users\Andrew\Desktop\DumpIt\WIN-TK0PBVQLQNB-20120719-065010.raw
```

   You can also use the consoles plugin:

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 consoles
Volatile Systems Volatility Framework 2.3_beta
****
ConsoleProcess: conhost.exe Pid: 2308
Console: 0xff0f6200 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Users\Andrew\Desktop\DumpIt\DumpIt.exe
Title: C:\Users\Andrew\Desktop\DumpIt\DumpIt.exe
AttachedProcess: DumpIt.exe Pid: 1836 Handle: 0x60

CommandHistory: 0x1feb70 Application: DumpIt.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60

Screen 0x1e0fe0 X:80 Y:300
Dump:
DumpIt - v1.3.2.20110401 - One click memory memory dumper
Copyright (c) 2007 - 2011, Matthieu Suiche <http://www.msuiche.net>
Copyright (c) 2010 - 2011, MoonSols <http://www.moonsols.com>

Address space size: 268435456 bytes (256 Mb)
Free space size: 10364592128 bytes (9884 Mb)

*** Destination = \??\C:\Users\Andrew\Desktop\DumpIt\WIN-TK0PBVQLQNB-20120719-065010.raw**

--> Are you sure you want to continue? [y/n] y
+ Processing...

4. What was the virtual offset of the usbccgp.sys driver?

$ volatility -f hands-on-plugins.raw --profile=Win7SP1x64 modules |grep -i usbccgp.sys
0xfffffa80058f7820 usbccgp.sys 0xfffff88004293000 0x1d000
\SystemRoot\system32\DRIVERS\usbccgp.sys

5. Which DLL files were loaded into the DumpIt process?
Since this is a 64bit system, we know that DLLs for 32bit processes are found in the VAD. We also know that DumpIt is a 32bit process by looking at the output from pslist or dlllist:

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 -p 1836 pslist
Volatile Systems Volatility Framework 2.3_beta

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0xfffffa8006843b30	DumpIt.exe	1836	1624	2	45	1	1	2012-07-19 06:50:07 UTC+0000	

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 -p 1836 dlllist
Volatile Systems Volatility Framework 2.3_beta
****
DumpIt.exe pid: 1836

```
Command line : "C:\Users\Andrew\Desktop\DumpIt\DumpIt.exe"
Note: use ldrmodules for listing DLLs in Wow64 processes
Base                             Size         LoadCount Path
-------------------- ---------- ------------------ ----
0x0000000000f50000         0x35000          0xffff C:\Users\Andrew\Desktop\DumpIt\DumpIt.exe
0x0000000077040000         0x1a9000          0xffff C:\Windows\SYSTEM32\ntdll.dll
0x0000000073540000         0x3f000            0x3 C:\Windows\SYSTEM32\wow64.dll
0x00000000734e0000         0x5c000            0x1 C:\Windows\SYSTEM32\wow64win.dll
0x00000000734d0000         0x8000             0x1 C:\Windows\SYSTEM32\wow64cpu.dll
```

We can use ldrmodules to see the full list of DLLs:

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 -p 1836 ldrmodules

```
Volatile Systems Volatility Framework 2.3_beta
Pid      Process            Base               InLoad InInit InMem MappedPath
-------- -------------------- ------------------ ------ ------ ----- ----------
   1836 DumpIt.exe         0x0000000074c10000 False  False  False \Windows\SysWOW64\kernel32.dll
   1836 DumpIt.exe         0x0000000000f50000 True   False  True  \Users\Andrew\Desktop\DumpIt\DumpIt.exe
   1836 DumpIt.exe         0x0000000076830000 False  False  False \Windows\SysWOW64\user32.dll
   1836 DumpIt.exe         0x0000000073540000 True   True   True  \Windows\System32\wow64.dll
   1836 DumpIt.exe         0x0000000077040000 True   True   True  \Windows\System32\ntdll.dll
   1836 DumpIt.exe         0x0000000074900000 False  False  False \Windows\SysWOW64\sspicli.dll
   1836 DumpIt.exe         0x0000000074f50000 False  False  False \Windows\SysWOW64\msvcrt.dll
   1836 DumpIt.exe         0x0000000075160000 False  False  False \Windows\SysWOW64\gdi32.dll
   1836 DumpIt.exe         0x0000000076740000 False  False  False \Windows\SysWOW64\lpk.dll
   1836 DumpIt.exe         0x0000000074a80000 False  False  False \Windows\SysWOW64\advapi32.dll
   1836 DumpIt.exe         0x0000000075390000 False  False  False \Windows\SysWOW64\rpcrt4.dll
   1836 DumpIt.exe         0x00000000766a0000 False  False  False \Windows\SysWOW64\usp10.dll
   1836 DumpIt.exe         0x00000000764a0000 False  False  False \Windows\SysWOW64\sechost.dll
   1836 DumpIt.exe         0x0000000077220000 False  False  False \Windows\SysWOW64\ntdll.dll
   1836 DumpIt.exe         0x0000000074d20000 False  False  False \Windows\SysWOW64\shlwapi.dll
   1836 DumpIt.exe         0x00000000734d0000 True   True   True  \Windows\System32\wow64cpu.dll
   1836 DumpIt.exe         0x0000000074b20000 False  False  False \Windows\SysWOW64\imm32.dll
   1836 DumpIt.exe         0x00000000756e0000 False  False  False \Windows\SysWOW64\msctf.dll
   1836 DumpIt.exe         0x00000000748f0000 False  False  False \Windows\SysWOW64\cryptbase.dll
   1836 DumpIt.exe         0x0000000076500000 False  False  False \Windows\SysWOW64\KernelBase.dll
   1836 DumpIt.exe         0x00000000734e0000 True   True   True  \Windows\System32\wow64win.dll
```

6. What file is mapped from the VAD starting at 0xfffffa80054a6b50 in the SpotifyWebHelp process?

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 pslist |grep -i SpotifyWebHelp

```
Volatile Systems Volatility Framework 2.3_beta
0xfffffa8006565060 SpotifyWebHelp      1280   1624      5      145      1      1 2012-07-19 06:47:19 UTC+0000
```

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 vadinfo -p 1280 |less

Once in less we can type the following to search for the offset:
/0xfffffa80054a6b50

VAD node @ **0xfffffa80054a6b50** Start 0x0000000076410000 End 0x000000007649efff Tag Vad

Flags: CommitCharge: 3, Protection: 7, VadType: 2
Protection: PAGE_EXECUTE_WRITECOPY
Vad Type: VadImageMap
ControlArea @fffffa80058b3a60 Segment fffff8a00424baf0
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences: 1 NumberOfPfnReferences: 2
NumberOfMappedViews: 2 NumberOfUserReferences: 3
WaitingForDeletion Event: 00000000
Control Flags: Accessed: 1, File: 1, Image: 1
FileObject @fffffa80058ad8a0, Name: **\Windows\SysWOW64\oleaut32.dll**
First prototype PTE: fffff8a00424bb38 Last contiguous PTE: fffffffffffffffc
Flags2: Inherit: 1

7. What port is lsass listening on?

$ volatility –f hands-on-plugins.raw --profile=Win7SP1x64 netscan |grep -i lsass
Volatile Systems Volatility Framework 2.3_beta
0xb161ad0 TCPv4 0.0.0.0:**49154** 0.0.0.0:0 LISTENING 508 lsass.exe
0xb161ad0 TCPv6 :::**49154** :::0 LISTENING 508 lsass.exe
0xc25e150 TCPv4 0.0.0.0:**49154** 0.0.0.0:0 LISTENING 508 lsass.exe

Memory – Keylogger - Questions

You have been given a memory image of a system that was believed to be infected with a key logger. You must answer the following questions to determine the activity of the key logger and its potential effects on the user.

Profile: Win7SP1x64

Suggested Plugins: psxview, handles, printkey, procdump, dumpfiles

1. Which process is suspicious and why?

2. Which file were keystrokes logged to?

3. What persistence mechanism was used by the keylogger?

4. What data was recorded by the keylogger on the infected computer?

Plugin Hints (****Spoilers!*****):

1. Run psxview and look for 'False' columns

2. Run handles against the suspicious process, since it is hidden you must use the -o flag to specify the physical address of the process. DO NOT USE –p as it will not work.

3. Run handles again, but this time looking for registry keys.

4. Use dumpfiles to extract the keylogger log file. Is the data encoded? If so, can you decode it?

Memory – Keylogger - Answers

1. Which process is suspicious and why?

Answer: mswinnt.exe PID: 1300

If you run **pslist**, you will see that there are no processes out of the ordinary. If you run **psscan**, you will see that the mswinnt process appears without an exit time, but that it does not appear in **pslist**. This is a good indication that a process has broken itself from the active process list. We can confirm this by using **psxview** and see that mswinnt is in all places except **pslist**:

```
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw psxview
Offset(P)  Name           PID  pslist psscan thrdproc pspcid csrss session deskthrd
---------- -------------- ---- ------ ------ -------- ------ ----- ------- ------
0x3f782b30  iexplore.exe  628  True   True   True     True   True  True    True
0x3e7e4710 svchost.exe    980  True   True   True     True   True  True    True
0x3f7c8960  mswinnt.exe  1300  False  True   True     True   True  True    True
0x3eb65b30 lsass.exe      496  True   True   True     True   True  True    False
```

Since the process does not appear in the process list, we will need to use the '-o' parameter with its physical address (0x3f7c8960) in order to examine it.

2. Which file were keystrokes logged to?

We can answer this question a number of ways.

a) Handles

We can look at the opened file handles of the process and see if any point to the keylogger:

```
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw handles -o 0x3f7c8960 -t
File
Volatility Foundation Volatility Framework 2.4
Offset(V)             Pid             Handle             Access Type
Details
------------------ ------ ------------------ ------------------ -------------
--- -------
0xfffffa800e82e2d0   1300               0x10           0x100020 File
\Device\HarddiskVolume1\Windows
0xfffffa800ea02440   1300               0x1c           0x100020 File
\Device\HarddiskVolume1\Users\Andrew\Desktop
0xfffffa800d166500   1300               0x64           0x120196 File
\Device\HarddiskVolume1\Users\Andrew\Desktop\log.txt
```

The “log.txt” file on the desktop seems like a plausible candidate and is indeed the log file. We can use the next two approaches to prove this.

Note: The reason the log file appears in handles output is because the keylogger opens the log file at startup and then never close it. There are other malware samples that, for each entry to be logged, open, write to, and then close the file. For these types of samples, the log file would likely not appear in handles output unless the memory capture tool happened to be active while the malware was writing. However, if you used the **filescan** command, which finds residual, potentially historically used file objects, you may see it.

b) Command line arguments

If we read the command line arguments to the process:

```
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw dlllist -o 0x3f7c8960
Volatility Foundation Volatility Framework 2.4
************************************************************
mswinnt.exe pid:   1300
Command line : mswinnt.exe  --logfile=log.txt --encryption-index=5
Note: use ldrmodules for listing DLLs in Wow64 processes
<snip>
```

We see that it takes two parameters. One is –logfile, which “log.txt” is the same as we saw in handles output. This further strengthens our assumption that log.txt is the keylogger output file. We also see –encryption-index=5, which will be important in another question.

c) Registry

This is explained in the answer to the next question.

3. What persistence mechanism was used by the keylogger?

If we look at the registry keys opened by the process, we see that it has a handle open to the run key:

```
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw handles -o 0x3f7c8960 -t
Key
Volatility Foundation Volatility Framework 2.4
Offset(V)          Pid             Handle             Access Type
Details
------------------ ------ ------------------ ------------------ -------------
--- -------
<snip>
```

```
0xfffff8a002030ad0   1300                0x60            0xf003f Key
USER\S-1-5-21-1133905431-3037184594-10822689-
1000\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN
0xfffff8a0021296c0   1300                0x68               0x1 Key
MACHINE\SYSTEM\CONTROLSET001\CONTROL\NLS\CUSTOMLOCALE
```

If we print this with printkey, we see that the malware writes itself with the given command line parameters to the user's startup key:

```
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw printkey -K
"SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN"
Volatility Foundation Volatility Framework 2.4
Legend: (S) = Stable   (V) = Volatile

---------------------------
Registry: \??\C:\Users\Andrew\ntuser.dat
Key name: Run (S)
Last updated: 2013-03-10 22:47:09 UTC+0000

Subkeys:

Values:
REG_SZ        mswinnt         : (S) "C:\Users\Andrew\Desktop\mswinnt.exe" --
logfile=log.txt --encryption-index=4
```

4. What data was recorded by the keylogger on the infected computer?

To find the captured data, we can use dumpfiles as well as the handles plugin. If we run handles with the "-P" flag set, then it will print the physical offset of the _FILE_OBJECT for the file:

```
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw handles -o
0x3f7c8960 -t File -p 1300 -P | grep log.txt
Volatility Foundation Volatility Framework 2.4
0x000000003fb66500   1300                0x64           0x120196 File
\Device\HarddiskVolume1\Users\Andrew\Desktop\log.txt

$ mkdir log.txt-dump
$ volatility --profile=Win7SP1x64 -f keylogger-capture.raw dumpfiles -Q
0x3fb66500 -D log.txt-dump
Volatility Foundation Volatility Framework 2.4
DataSectionObject 0x3fb66500   None
\Device\HarddiskVolume1\Users\Andrew\Desktop\log.txt
SharedCacheMap 0x3fb66500   None
\Device\HarddiskVolume1\Users\Andrew\Desktop\log.txt
$ cat log.txt-dump/file.None.0xfffffa800cddfb00.vacb
<?xml version="1.0"?>
```

```
<entry><time>3/10/2013 6:25:35
PM</time><keys>b3RqODlvdGo4OTFoc3MuaHRyIChXanl6d3MpIChHZnNwIHRrIEZyanduaGYgfC
BUc3Fuc2ogR2ZzcG5zbCB8IFhubHMgTnMgfCBUc3Fuc2ogTkkgLSBCbik=</keys></entry>
<entry><time>3/10/2013 6:28:49
PM</time><keys>Z3RnMTIzZ3RnMXU0eHgyMHdpd2ppaW55Lmh0ci8vdy9zanl4amggKFdqeXp3cy
kgKFJmc2ZsaiBHZnNwbnNsICYgUWpzaW5zbCBGaGh0enN5eCB8IEhmdW55ZnEgVHNqIFRzcW5zaiB
HZnNwKQ==</keys></entry>
```

In this output we see an XML header followed by XML two entries. If we base64 decode the first xml entry we see what looks like jumbled text:

```
$ echo
b3RqODlvdGo4OTFoc3MuaHRyIChXanl6d3MpIChHZnNwIHRrIEZyanduaGYgfCBUc3Fuc2
ogR2ZzcG5zbCB8IFhubHMgTnMgfCBUc3Fuc2ogTkkgLSBCbik= | base64 -d
otj89otj891hss.htr (Wjyzws) (Gfsp tk Frjwnhf | Tsqnsj Gfspnsl | Xnls
Ns | Tsqnsj NI - Bn)
```

This looks like either a classic Caesar shift or substitution cipher. Since we know that an encryption-index was passed on the command line, we can use this to attempt to decode with a shift of 5. This reveals the following text:

```
joe89joe891cnn.com (Return) (Bank of America | Online Banking | Sign
In | Online ID - Wi)
```

This shows “joe89” being entered on Bank of America’s website. This is likely the person’s username. If we decrypt the second entry, we get:

```
bob123bob1p4ss20rdreddit.com//r/netsec (Return) (Manage Banking &
Lending Accounts | Capital One Online Bank)
```

Which s a username of bob123 and a password of ‘p4ss20rd’ on Capital One’s website.

Memory – Code Injection – Questions

You have been given a compressed memory sample infected with malware that hooks many APIs in processes. First, decompress the sample. Then, in order to determine the functionality of the malware, you must locate, isolate, and analyze it. The following questions will guide you through this process.

Profile: WinXPSP2x86

Suggested Plugins: malfind, apihooks, dumpfiles, filescan, pslist

Questions

1. The malware is known to hook APIs in Internet Explorer. Does the *malfind* plugin show signs of such activity in the Internet Explorer instance that is active in the memory sample?

2. Run *malfind* against the Internet Explorer process with the –D flag set to an output directory. What can you gather from static (strings) analysis of these two files?

3. Look at the full output of *pslist*. Do any of the process seem suspicious and related to strings associated with the extracted malware segments?

4. Run the *cmdline* plugin against the suspicious process to determine its location on disk. Then run the *filescan* plugin and search for files in this directory. Do you see any other components of the malware in this folder? Extract them with *dumpfiles* if so.

5. One of the extracted files appears to be a configuration file. Based on its contents, what does the malware attempt hide from the live system?

6. Run the *apihooks* plugin against the IE process and redirect the output to a file. Be sure to set the --quick flag before running the plugin. Which APIs has the malware hooked, and what benefits could be gained by hooking each API? For any that are unfamiliar, search them on MSDN to gather ideas on why malware might hook the particular API.

7. Pick three addresses listed as a "Hook address" in the *apihooks* output. Use *vadinfo* with the -*a* flag to map them back to their containing VAD. Do all of the hooks map to the same VAD? Does this VAD match what *malfind* automatically found and extracted?

Memory – Code Injection – Answers

1. The malware is known to hook APIs in Internet Explorer. Does the *malfind* plugin show signs of such activity in the Internet Explorer instance that is active in the memory sample?

Gathering of this information could be accomplished with the following Volatility invocation:

```
# volatility -f codeinjectionlab.mem malfind -p 344
Process: IEXPLORE.EXE Pid: 344 Address: 0x180000
Vad Tag: Vad  Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x00180000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00      MZ..............
0x00180010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x00180020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00180030  00 00 00 00 00 00 00 00 00 00 00 00 d8 00 00 00   ................

0x00180000 4d              DEC EBP
0x00180001 5a              POP EDX
0x00180002 90              NOP
0x00180003 0003            ADD [EBX], AL

Process: IEXPLORE.EXE Pid: 344 Address: 0x7ffa0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 5, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x7ffa0000  e8 00 00 00 00 58 2d b6 5d 40 00 c3 5f 2e 2d 3d   .....X-.]@.._.-=
0x7ffa0010  5b 48 61 63 6b 65 72 20 44 65 66 65 6e 64 65 72   [Hacker.Defender
0x7ffa0020  5d 3d 2d 2e 5f 00 00 00 00 00 00 00 00 04 00 00   ]=-._...........
0x7ffa0030  00 6b 65 72 6e 65 6c 33 32 2e 64 6c 6c 00 53 65   .kernel32.dll.Se

0x7ffa0000 e800000000  CALL 0x7ffa0005
0x7ffa0005 58          POP EAX
0x7ffa0006 2db65d4000  SUB EAX, 0x405db6
0x7ffa000b c3          RET
```

This would have produced two suspicious memory regions. The first starts with MZ and is likely an injected DLL. The other has interesting strings related to hacking tools. Combined, the output definitely warrants further investigation.

2. Run *malfind* against the Internet Explorer process with the –D flag set to an output directory.

To generate the output with *malfind*, Volatility could have been run as follows:

```
# volatility -f codeinjectionlab.mem malfind -p 344 –D output
```

This should produce two files:

```
# ls output
process.0x821fe020.0x180000.dmp  process.0x821fe020.0x7ffa0000.dmp
```

strings based analysis of these files finds references to many Windows APIs commonly used for code injection, process monitoring, and other misuse purposes. The output also contained the following string, which gives a strong hint on which malware is present:

```
_.-=[Hacker Defender]=-._
```

Hacker Defender is a well-known rootkit. Furthermore, several references to mailslots can be found in the strings output:

```
\\.\mailslot\hxdef-rk100s30C4E28D
\\.\mailslot\hxdef-rkc000
\\.\mailslot\hxdef-rkb000
\Device\Mailslot\hxdef*
```

3. Look at the full output of *pslist*.

In the *pslist* output is a process whose name has a similar pattern to the mailslots and other strings found in the previous question:

```
0x81cff8c0 hxdef100.exe          3348    692      2       31      0      0 2017-11-22 05:36:51 UTC+0000
```

4. Run the *cmdline* plugin against the suspicious process to determine its location on disk.

```
# volatility  -f codeinjectionlab.mem cmdline -p 3348
Volatility Foundation Volatility Framework 2.5
************************************************************************
hxdef100.exe pid:   3348
Command line : "C:\Documents and Settings\Administrator\Desktop\hxdef100.exe"
```

This output shows the malware running from the Administrator's Desktop folder. Searching *filescan* output for this folder produces the following results:

```
# volatility  -f codeinjectionlab.mem filescan | grep "Administrator.Desktop"
0x223fb10      1      0 R--r-d \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\hxdef100.exe
0x22f7bb8      1      1 R--rw- \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop
0x22feba0      3      1 R--rwd \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop
0x23c0188      1      0 R--r-d \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\hxdefdrv.sys
0x250cae8      1      0 R--r-- \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\hxdef100.ini
0x250ccf0      1      0 -WD--- \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\hxdef100.exe
```

In particular, we find references to the running malware itself along with a kernel driver and a .ini file with very similar base names as the malware.

5. One of the extracted files appears to be a configuration file. Based on its contents, what does the malware attempt hide from the live system?

To extract the .ini file, you can use the previous *filescan* output (namely the physical offset in the first column) with the –Q flag to *dumpfiles*:

```
# volatility  -f codeinjectionlab.mem dumpfiles -D output -Q 0x000000000250cae8 -n
Volatility Foundation Volatility Framework 2.5
DataSectionObject 0x0250cae8   None   \Device\HarddiskVolume1\Documents and
Settings\Administrator\Desktop\hxdef100.ini
# ls output
file.None.0x82106770.hxdef100.dat
# cat output/file.None.0x82106770.hxdef100.dat
 [H<<<idden T>>a/"ble]
>h"xdef"*
r|c<md\.ex<e::
calc.exe
"[:\:R:o:o\:t: :P:r>:o:c<:e:s:s:e<:s:>]
h<x>d<e>:f<*
<\r\c:\m\d.\e\x\e
Ha>:ck"er//Def\ender*
<snip>
```

In the abbreviated output, it can be seen that the rootkit is configured to hide all strings related to the "hxdef" prefix as well as calc.exe.

6. Run the *apihooks* plugin against the IE process and redirect the output to a file.

Running the apihooks plugin against the IE process shows that 31 functions are hooked:

```
# volatility -f codeinjectionlab.mem apihooks -p 344 --quick > apihooks.344
# grep -c Function: apihooks.344
31
```

Review of these functions show they can control APIs related to networking, library loading, debugging, reading and writing the registry, enumerating services, and more.

7. Pick 3 addresses listed as the hook address in the *apihooks* output.

The following shows this process for two hook addresses. As can be seen, both map the malicious region reported by malfind:

```
# volatility -f codeinjectionlab.mem vadinfo -p 344 -a 0x7ffa488d
Pid:    344
VAD node @ 0x81e8aef0 Start 0x7ffa0000 End 0x7ffa4fff Tag VadS
Flags: CommitCharge: 5, MemCommit: 1, PrivateMemory: 1, Protection: 6
Protection: PAGE_EXECUTE_READWRITE

# volatility -f codeinjectionlab.mem vadinfo -p 344 -a 0x7ffa4828
Pid:    344
VAD node @ 0x81e8aef0 Start 0x7ffa0000 End 0x7ffa4fff Tag VadS
Flags: CommitCharge: 5, MemCommit: 1, PrivateMemory: 1, Protection: 6
Protection: PAGE_EXECUTE_READWRITE
```

Memory - Taking on the Powershell Empire - Questions

Profile: Win7SP1x86

Suggested plugins: malfind, netscan, pslist, pstree, cmdline, dlllist, volshell

Suggested helper tools: strings / WinHex

Questions

1. Examine the network connections. Which processes look out of the ordinary?
2. What was the malicious file (infection vector)? Hint: Further examine the processes you found in question 1.
3. Which process is hosting the powershell empire? Hint: Look at parent/child relationships
4. What was the tool used (in addition to powershell empire) to hack the machine?
5. Recover as many powershell commands as you are able to from the memory sample. Hint: a lot of them contain the strings –NoP, -sta, -NonI, -Enc. Decode the encoded commands.
6. Were credentials dumped?

Memory – Taking on the Powershell Empire - Answers

1) Examine the network connections. Which processes look out of the ordinary?

In this case, there aren't a lot of network connections to choose from, but we'll quickly notice three that stand out:

```
$ vol.py --profile=Win7SP1x86 -f power.vmem netscan
[snip]
0x1d8dd0c0 TCPv4 172.16.99.146:49385 172.16.99.179:8080 CLOSED 3040     rundll32.exe
[snip]

0x1f978340 TCPv4 172.16.99.146:49160 172.16.99.179:4444 ESTABLISHED 2128
Campus_IT_Poli

0x1f9f0830 TCPv4 172.16.99.146:49163 172.16.99.179:4444 ESTABLISHED 3696 rundll32.exe
```

We'll need to take a closer look at these three processes.

2) What was the malicious file (infection vector)?

If we look at the verbose command lines, we'll see the malicious document immediately:

```
$ vol.py --profile=Win7SP1x86 -f power.vmem cmdline -v -p 2128,3696,3040
Volatility Foundation Volatility Framework 2.5
************************************************************************
\Device\HarddiskVolume1\Users\user\Desktop\Campus_IT_Policy_orig.pdf pid:   2128
************************************************************************
\Device\HarddiskVolume1\Windows\System32\rundll32.exe pid:   3696
Command line : rundll32.exe
************************************************************************
\Device\HarddiskVolume1\Windows\System32\rundll32.exe pid:   3040
Command line : rundll32.exe
```

3) Which process is hosting the powershell empire?

If we use the **pstree** plugin, we'll see that all of the powershell processes are spawned by a particular process- rundll32.exe with PID 3040:

```
$ vol.py --profile=Win7SP1x86 -f power.vmem pstree
0x8434e030:rundll32.exe     3040   3308     15    440 2016-04-27 21:46:15 UTC+0000
. 0x843604c0:powershell.exe 2840   3040      0 ------ 2016-04-27 22:01:55 UTC+0000
. 0x84378030:powershell.exe 2568   3040      7    249 2016-04-27 21:57:29 UTC+0000
. 0x8433cd40:powershell.exe 1980   3040      7    246 2016-04-27 21:57:18 UTC+0000
. 0x8612bc48:powershell.exe 3920   3040      6    242 2016-04-27 21:54:15 UTC+0000
```

This process also was the only one with a different connection port than the other two malicious processes from earlier.

4) What was the tool used (in addition to powershell empire) to hack the machine?

Metasploit.

The connection ports for two of the processes are the default for meterpreter (4444).

Also, if we use the **malfind** plugin, we will find a few injected DLLs and we can dump those out and then check out the exports using **strings**:

```
$ vol.py --profile=Win7SP1x86 -f power.vmem malfind
[snip]
Process: rundll32.exe Pid: 3696 Address: 0x4e0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 99, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x004e0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x004e0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x004e0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x004e0030  00 00 00 00 00 00 00 00 00 00 00 00 18 01 00 00   ................

0x004e0000 4d              DEC EBP
0x004e0001 5a              POP EDX
0x004e0002 90              NOP
0x004e0003 0003            ADD [EBX], AL
0x004e0005 0000            ADD [EAX], AL
0x004e0007 000400          ADD [EAX+EAX], AL
0x004e000a 0000            ADD [EAX], AL
0x004e000c ff              DB 0xff
0x004e000d ff00            INC DWORD [EAX]

$ vol.py --profile=Win7SP1x86 -f power.vmem dlldump -D powerout/ -p 3696 -b 0x4e0000
Volatility Foundation Volatility Framework 2.5
Process(V) Name                 Module Base Module Name          Result
---------- -------------------- ----------- -------------------- ------
0x843e6940 rundll32.exe         0x0004e0000 UNKNOWN              OK:
module.3696.1f9e6940.4e0000.dll
```

If you run strings on the dumped file then you will see many references to Metasploit capabilities. For example, you will see a repeated string of *ReflectiveLoader*, which if you Google search, will bring many web pages related to Metasploit.

5) Recover as many powershell commands as you are able to from the memory sample. Hint: a lot of them contain `-NoP -sta -NonI -W Hidden -Enc`. Decode the encoded commands

Searching for powershell related data in strings would have found a number of Base64 encoded commands, such as:

"C:\Windows\System32\WindowsPowershell\v1.0\powershell.exe" -NoP -sta -NonI -W Hidden -Enc G8AdwApACAALQBnAHQAIAAkAHMAdABhAHIAdAB0AGkAbQBlAC4A <snip>

You can base64 these on the Linux command line with:

$ echo “base64 encoded string” | base64 -d

6) Were credentials dumped and if so?

Yes, they were, and the mimikatz powershell empire module was used. We can see evidence of this by looking at the malfind output:

```
Process: lsass.exe Pid: 508 Address: 0x190000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00190000  5c 39 24 76 d3 33 24 76 d0 d9 23 76 21 3e 23 76   \9$v.3$v..#v!>#v
0x00190010  cc bc 23 76 7c ca 23 76 90 ba 23 76 73 61 6d 73   ..#v|.#v..#vsams
0x00190020  72 76 2e 64 6c 6c 00 53 61 6d 49 43 6f 6e 6e 65   rv.dll.SamIConne
0x00190030  63 74 00 53 61 6d 72 4f 70 65 6e 44 6f 6d 61 69   ct.SamrOpenDomai

0x00190000 5c              POP ESP
0x00190001 392476          CMP [ESI+ESI*2], ESP
0x00190004 d333            SAL [EBX], CL
0x00190006 2476            AND AL, 0x76
0x00190008 d0d9            RCR CL, 0x1
0x0019000a 237621          AND ESI, [ESI+0x21]
0x0019000d 3e2376cc        AND ESI, [ESI-0x34]
[snip]
```

Looking at strings of the extracted region would have confirmed this.

Memory – Analysis with Yara – Questions

For this exercise, you have been given 6 memory samples. In order to determine which samples are infected with specific pieces of malware (more than one sample can be infected with the same malware), you must write custom Yara rules given IOCs that will positively and uniquely match the malware samples.

The following table lists each sample along with its Volatility profile:

1.mem	WinXPSP2x86
2.mem	WinXPSP2x86
3.mem	WinXPSP2x86
4.mem	WinXPSP2x86
5.mem	Win10x64_15063 (kdbg: 0xf800245fb8dc)
6.mem	WinXPSP2x86

Note: At least one of these rules targets kernel level malware, so you must scan both process and kernel memory, meaning run each rule twice, once with -K set to Volatility and once without.

As an example, to run completely against 1.mem, you would do the following in two separate commands:

```
$ volatility -f 1.mem --profile=WinXPSP2x86 yarascan -y $RULE

$ volatility -f 1.mem --profile=WinXPSP2x86 yarascan -y $RULE -K
```

You would substitute $RULE with the path to your custom Yara rules files.

The criteria for the Yara rules are as follows:

The rule for malware 1 must match all of the following:

- The Unicode string "Realtek Semiconductor Corp"
- The Unicode string "SeLoadDriverPrivilege"

The rule for malware 2 must match all of the following:

- The Unicode OR ASCII string “1B372133-BFFA-4dba-9CCF-5474BED6A9F6”

The rule for malware 3 must match all of the following:

- The Unicode OR ASCII string “bak.cosmogonid.com”

After creating the rules, run them against each memory sample until a positive hit is found.

Hint: You can open multiple terminals (or terminal tabs) at once in your VM and run Volatility many times concurrently. This will greatly speed up your analysis.

Bonus: After you have successfully created a Yara rule for each malware sample, go back and search the indicator(s) online in order to determine which malware you detected.

Memory – Analysis with Yara – Answers

For the first Yara rule, the following would have met all the criteria and triggered on 3.mem:

```
rule Stuxnet {
        strings:
                $a = "Realtek Semiconductor Corp" wide
                $b = "SeLoadDriverPrivilege" wide
        condition:
                all of them
}
```

The second rule would trigger on 2.mem:

```
 rule ZeroAccess {
        strings:
                $a = "1B372133-BFFA-4dba-9CCF-5474BED6A9F6" wide ascii
        condition:
                $a
 }
```

The third rule on 6.mem:

```
rule PlugX {
        strings:
                $a = "bak.cosmogonid.com" wide ascii
        condition:
                $a
 }
```

Notes on automation:

Since you were given the profile, a simple bash script like the following could be used to auto-run a single Yara rule file against all the samples in both process and kernel memory. It takes the path of the Yara rule as the first argument, and greatly speeds up testing across all samples.

```
#!/bin/bash -v

VOLATILITY='python /home/training/volatility/vol.py'
RULE=$1

$VOLATILITY -f 1.mem --profile=WinXPSP2x86 yarascan -y $RULE
$VOLATILITY -f 1.mem --profile=WinXPSP2x86 yarascan -y $RULE  -K

$VOLATILITY -f 2.mem --profile=WinXPSP2x86 yarascan -y $RULE
$VOLATILITY -f 2.mem --profile=WinXPSP2x86 yarascan -y $RULE  -K

$VOLATILITY -f 3.mem --profile=WinXPSP2x86 yarascan -y $RULE
```

```
$VOLATILITY -f 3.mem --profile=WinXPSP2x86 yarascan -y $RULE  -K

$VOLATILITY -f 4.mem --profile=WinXPSP2x86 yarascan -y $RULE  -K
$VOLATILITY -f 4.mem --profile=WinXPSP2x86 yarascan -y $RULE

$VOLATILITY -f 6.mem --profile=WinXPSP2x86 yarascan -y $RULE  -K
$VOLATILITY -f 6.mem --profile=WinXPSP2x86 yarascan -y $RULE

$VOLATILITY -f 5.mem --profile=Win10x64_15063 --kdbg=0xf800245fb8dc yarascan -y $RULE -K
$VOLATILITY -f 5.mem --profile=Win10x64_15063 --kdbg=0xf800245fb8dc yarascan -y $RULE
```