Advanced artifact analysis

Artifact analysis training material

November 2014

European Union Agency for Network and Information Security www.enisa.europa.eu

About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

Authors

This document was created by Lauri Palkmets, Cosmin Ciobanu, Yonas Leguesse, and Christos Sidiropoulos in consultation with DFN-CERT Services[1] (Germany), ComCERT[2] (Poland), and S-CURE[3] (The Netherlands).

Contact

For contacting the authors please use cert-relations@enisa.europa.eu

For media enquires about this paper, please use press@enisa.europa.eu

Acknowledgements

ENISA wants to thank all institutions and persons who contributed to this document. A special 'Thank You' goes to the following contributors:

1. Todor Dragostinov from ESMIS, Bulgaria
2. Michael Ligh from MNIN.ORG, United States

[1] Klaus Möller, and Mirko Wollenberg
[2] Mirosław Maj, Tomasz Chlebowski, Krystian Kochanowski, Dawid Osojca, Paweł Weżgowiec, and Adam Ziaja
[3] Michael Potter, Alan Robinson, and Don Stikvoort

Legal notice

Copyright Notice

Table of Contents

Main Objective	During the course of the exercise the students will learn how to obtain memory images from different sources and to analyse them. Both Windows and Linux systems will be covered.	
Targeted Audience	CERT staff involved with the technical analysis of incidents, especially those dealing with forensics. In-depth knowledge of operating system concepts is a prerequisite. Python knowledge is helpful but not required.	
Total Duration	6 – 7.5 hours	
Time Schedule	**Introduction** to the exercise (Memory forensics)	1.5 hours
	Task 1: Acquiring memory images	1 hour
	Task 2: Basic Windows memory analysis	1-1.5 hours
	Task 3: Basic Linux memory analysis	1-1.5 hours
	Task 4: Obtaining and analysing malware from memory	1-1.5 hours
	Summary of the exercise	0.5 hours
Frequency	Once per person	

1 General description

During the course of this exercise, the students will learn how to acquire and analyse memory images from Windows and Linux operating systems with the Volatility toolkit[4]. Students that have completed the "Digital forensics"[5] course may already be familiar with some of the concepts covered in this exercise.

At the beginning, the trainer will introduce the basic concepts of memory forensics, such as acquisition of memory and its analysis. Additionally, the students will be given basic information on how to use Volatility.[4]

In the first part, the students will learn how to acquire memory images from Windows and Linux operating systems.

In the second and third part, the students will perform basic analysis tasks while working with Windows and Linux memory dumps.

In the last part, the students are confronted with advanced analysis techniques, such as identifying and isolating a malware sample from a given memory image.

The exercise can be held with all parts in sequence. But it is also possible to hold only the introduction, together with either the Windows or the Linux parts to focus on only one operating system or if the time frame doesn't allow for a full training, additionally there will also be an open discussion at the end of the exercise.

1.1 Prerequisites

It is assumed that the students are familiar with

- The general process of incident response and digital forensics.[6,7,8,9]
- Basic concepts from C / Assembler programming such as low level data types (char, int), arrays, structures, etc.[10,11,12,13,14]
- General operating system concepts such as processes, threads, stack, heap, kernel, memory management, etc.[15,16,17,18]

[4] http://www.volatilityfoundation.org/ (last visited: 30. 9. 2014)
[5] https://www.enisa.europa.eu/activities/cert/support/exercise, exercise no 24 (last visited: 30. 9. 2014)
[6] https://www.enisa.europa.eu/activities/cert/background (last visited: 30. 9. 2014)
[7] https://www.enisa.europa.eu/activities/cert/support (last visited: 30. 9. 2014)
[8] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf (last visited: 30. 9. 2014)
[9] http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf (last visited: 30. 9. 2014)
[10] Mitchell: "Concepts in Programming Languages", Cambrigde University Press, ISBN 0521780985
[11] Van Roy, et al.: "Concepts, Techniques, and Models of Computer Programming", MIT Press, ISBN: 9780262220699
[12] http://www.cprogramming.com/(last visited: 30. 9. 2014)
[13] http://en.wikibooks.org/wiki/C_Programming (last visited: 30. 9. 2014)
[14] http://en.wikibooks.org/wiki/X86_Disassembly (last visited: 30. 9. 2014)
[15] Silberschatz, et al.: "Operating System Concepts", 9th ed., Wiley, ISBN-13: 978-1118063330
[16] Russinovich, et al.: "Windows Internals", 6th ed., Microsoft Press, ISBN-13: 978-0735648739 (part1), ISBN-13: 978-0735665873 (part 2), http://technet.microsoft.com/de-de/sysinternals/bb963901.aspx (last visited: 30. 9. 2014)
[17] Wolfgang Mauerer: "Professional Linux Kernel Architecture",Wrox, ISBN-13: 978-0470343432
[18] Bovet, et al.: "Understanding the Linux Kernel", 3rd ed., O'Reilly, ISBN-13: 978-0596005658

- Standard Unix command line tools, such as **grep, sort,** etc.

1.2 Tools overview

The following section gives an overview about the tools used in this exercise.

1.2.1 Volatility

Volatility is an open source framework for the extraction of digital artifacts from volatile memory (RAM) samples. It is now (2014) developed and supported by The Volatility Foundation.[19]

1.2.2 Python for Volatility

Having the Python[20] interpreter and its libraries installed is a prerequisite to running Volatility. At least version 2.6 (better 2.7) is required. A Linux or Windows operating system with x86 or x64 architecture is preferred although Volatility should run on any system that supports Python. Python 3 is currently (as of version 2.3.1 and 2.4) not supported.

Furthermore, several Python libraries are needed by Volatility that typically are not installed automatically when installing the main Python package. Teachers should check for the presence of these packages when assembling their own analysis system.

1.2.3 PyCrypto

The Python Cryptography Toolkit[21] supplies hashing and encryption algorithms as well as encryption protocols and other routines, such as random number generators. It is needed by the lsadump and hashdump Volatility plug-ins.

1.2.4 PIL

The Python Imaging Library[22] supplies image manipulation routines for many different formats. This library is needed by the screenshots Volatility plug-in.

1.2.5 Distorm3

A disassembler library for x86/AMD64 architectures.[23]The library converts a binary data stream into assembler instructions, represented as python data structures. The library is needed by Volatility's apihooks, impscan, callbacks, volshell, linux_volshell and mac_volshell plug-ins.

1.2.6 Yara

A tool to classify malware, i. e. build malware signatures.[24] It is needed by the "yarascan" Volatility command.

The packages may be included as part of the Python package on the system or installed as a dependency when installing Volatility through a package management system (dep, yum, YaST, etc.). When installing manually, they need to be downloaded and installed prior to installing Volatility.

[19] http://www.volatilityfoundation.org/ (last visited: 30. 9. 2014)
[20] https://www.python.org/ (last visited 19. 8. 2014)
[21] https://www.dlitz.net/software/pycrypto (last visited 19.8. 2014)
[22] http://www.pythonware.com/products/pil (last visited 19.8. 2014)
[23] https://code.google.com/p/distorm (last visited 19.8. 2014)
[24] https://plusvic.github.io/yara/ (last visited 23. 9. 2014)

1.3 Memory acquisition tools

Although there are many tools for memory acquisition, the exercise will focus on two open source versions, one for Windows and one for Linux.

1.3.1 WinPmem, MacPmem

An open source memory acquisition tool with kernel driver signed by Microsoft.[25] It can be used on 32 and 64-bit versions of Windows, tools were developed by Michael Cohen (Google), but not included in the distribution. It must be downloaded separately from the Volatility source repository. This tool is needed for task one.

1.3.2 LiME

Linux Memory Extractor[26] – Memory acquisition for Linux/Android systems. It is not included in the distribution, and must be downloaded separately from Google code. It is needed in task one.

1.4 Other tools

The following tools are not strictly necessary to complete the exercise but they will ease some of the tasks considerably.

1.4.1 Strings

A tool to search for printable strings in binary files. Supplied with most Linux and Unix versions as part of the operating system or the GNU binutils[27] package. Needed in several tasks of the exercise. Windows users can use the GNU version together with MinGW[28] or Cygwin[29] or use the version from Sysinternals/Microsoft.[30]

1.4.2 Grep

Grep is a tool to search for strings matching regular expression in a set of files. It is available through the GNU grep[31] package. On Windows systems, the native **findstr** command can be used instead or the GNU version together with MinGW[32] or Cygwin.[33]

1.4.3 Ncat

Or any other netcat compatible tool. Needed for the transfer of images taken with WinPmem or LiME (or dd) over the network. The version used in this exercise is the one from the Nmap tool because it has a free Windows binary for download.[34]

Volatility and its dependencies are installed on the analysis virtual machine (Styx). WinPmem and Ncat are supplied as the “aq-tools” bundle that can be unpacked and directly used on Windows machines (they support both x86 and x64).

---

[25] https://sourceforge.net/projects/volatility.mirror/files/winpmem-1.4.1.zip/ (last visited 19.8. 2014)
[26] https://code.google.com/p/lime-forensics/ (last visited 19.8. 2014)
[27] https://www.gnu.org/software/binutils/ (last visited 19.8. 2014)
[28] http://www.mingw.org/ (last visited 19.8. 2014)
[29] https://www.cygwin.com/ (last visited 19.8. 2014)
[30] http://technet.microsoft.com/en-us/sysinternals/bb897439 (last visited 19.8. 2014)
[31] https://www.gnu.org/software/grep/ (last visited 19.8. 2014)
[32] http://www.mingw.org/ (last visited 19.8. 2014)
[33] https://www.cygwin.com/ (last visited 19.8. 2014)
[34] http://nmap.org/dist/ncat-portable-5.59BETA1.zip (last visited 19.8. 2014)

1.5 Alternatives

The following items are meant only as examples, not as a comprehensive list.

1.5.1 Kernel Debugger

Kernel Debuggers (KD for Windows, GDB for Linux or Mac OSX) can be used for some[35] tasks Volatility is used for.

- Often available for architectures and operating systems for which no forensic tools exist.
- Not limited to specific tasks or data structures like Volatility – therefore much more powerful.
- Requires debugging symbols together with the image to work. - will not work out-of-the box with raw images.
- User must have extensive knowledge about kernel data structures and programming.
- Tasks are not automatized and must be executed manually.

1.5.2 Mandiant Memoryze

Free but not open source tool from Mandiant[36]. For Windows and Max OS X memory analysis.

1.5.3 Second Look

From Raytheon Pikewerks[37]. For Linux memory analysis.

1.6 Virtual Memory images

The virtual memory images used in this exercise are from two sources:

The Linux images are from the Second Look website[38] and used with kind permission from Andrew Tappert and Jamie Hall.

- **Ubuntu-10.04.3-i386-LiveCD-clean.mem** - A clean image of a Ubuntu 10.04 system to be used as a baseline.
- **Ubuntu-10.04.3-i386--LiveCD-kbeast.mem** – An Ubuntu 10.04 system infected with the Kbeast kernel rootkit.
- **Windows-zeus.vmem** - A Windows XP system infected with the Zeus banking trojan.

The Windows image is from the Malware Cookbook[39] site and used with kind permission from Michael Ligh. All images are inside /home/enisa/enisa/ex4/memdumps/ and you can extract them with the unzip command.

1.7 Virtual Appliances

Two virtual appliances are used in this exercise.

[35] Many tasks that are exclusive to Volatility. For example kernel debugger cant show listening sockets or established connections, extract attacker command history, enumerate running services, etc.
[36] https://www.mandiant.com/resources/download/memoryze(last visited 19.8. 2014)
[37] http://secondlookforensics.com/ (last visited 19.8. 2014)
[38] https://secondlookforensics.com/linux-memory-images/ (last visited 26. 8. 2014)
[39] http://malwarecookbook.googlecode.com/svn-history/r26/trunk/17/1/zeus.vmem.zip (last visited 26. 8. 2014)

- **Styx** - A Ubuntu 14.04 system with the Volatility framework pre-installed. This will be used by the students to analyse the memory images. Login with user “enisa” and password “enisa”.
- **Opsu** - An OpenSuSE 12.3 system. This will be used in task 1.2 as a sample system to take a memory image from. Login with user “exercise” and password “exercise”.

The students are encouraged to bring their own Windows appliance with them for task 1.1. They can also use a full hardware-based Windows for the task. However, using a system with sensitive or critical data on it for the course is discouraged.

2 Exercise Course

2.1 Introduction

The teacher should use the introductory part for several purposes:

- A high level overview of digital memory forensics and digital memory analysis. References to the ENISA exercise 24 “Digital Forensics”[40] can be used here.
- The basic operation of the Volatility toolkit[41]. How Volatility is called from the command line, how images and profiles are specified and how help can be obtained using the –help and –info options.
- Basic concepts of how Volatility works internally. This should include the most important kernel data structures used by Volatility, like KPCR and KDEBUG and the way a kernel internally allocates memory, i.e. the Windows Object Manager[16] and the Linux slab allocator.[42,17] Based on this, the techniques of walking internal data structures and the basic principle of pool tag scanning should be introduced.[43]

2.2 Task 1: Acquiring memory images

In the introductory part, the teacher explains the concepts of address spaces,[44] vtypes and Volatility profiles. Also, the difference between “raw” and other memory dump types should be explained. Additionally, the acquisition methods used in this task(s) should be put into context with other methods of acquisition:

- Hardware-based (Firewire/Thunderbolt/ESATA/USB).[45]
- Virtual machine images/snapshots from VMware,[46] VirtualBox,[47] or other hypervisors.
- Software-based images. Aside from copy of /dev/mem, this also includes crashdumps[48] and hibernate files.[49]

In the last part, the teacher should explain the tools and commands used in the following task.

[40] https://www.enisa.europa.eu/activities/cert/support/exercise/ (exercise no 24) (last visited: 30. 9. 2014)
[41] https://github.com/volatilityfoundation/volatility/wiki/Volatility-Usage (last visited: 30. 9. 2014)
[42] http://en.wikipedia.org/wiki/Slab_allocation (last visited: 30. 9. 2014)
[43] http://computer.forensikblog.de/en/2009/04/scanning-for-file-objects.html (last visited: 30. 9. 2014)
[44] https://github.com/volatilityfoundation/volatility/wiki/Address-Spaces (last visited: 30. 9. 2014)
[45] https://github.com/volatilityfoundation/volatility/wiki/Firewire-Address-Space (last visited: 30. 9. 2014)
[46] https://github.com/volatilityfoundation/volatility/wiki/VMware-Snapshot-File (last visited: 30. 9. 2014)
[47] https://github.com/volatilityfoundation/volatility/wiki/Virtual-Box-Core-Dump (last visited: 30. 9. 2014)
[48] https://github.com/volatilityfoundation/volatility/wiki/Crash-Address-Space (last visited: 30. 9. 2014)
[49] https://github.com/volatilityfoundation/volatility/wiki/Hiber-Address-Space (last visited: 30. 9. 2014)

2.2.1 Task 1.1: Acquiring memory images from Windows

As a prerequisite, the students should have acquisition tools on their Windows system or appliance, or on a mapped network drive. The two following tools were included because they were open source, not due to specific features.

- **WinPmem** (or similar tool)
- **Netcat** or **Ncat** (from Nmap) (or similar tool)

The aq-tools folder supplied to the students contains a copy of **WinPmem** and **Ncat**.

The dump can be written to a file. As this will overwrite data on the local disk, it is advisable to do it to a removable drive or a network drive. If the in-memory data structures should be preserved as much as possible (and to avoid having malware infect the removable or network drive), the dump can be written to stdout and the forwarded with Ncat.

Commands used:

```
winpmem_1.4.exe testdump.raw                  # take the memory image on the Windows System
                                              # must be done as administrator
dir testdump.raw                              # show the memory image dump file

netcat -l -p 11111 > testdump.crash           # run on the receiving system (here: styx)
winpmem_1.4.exe -d - | ncat.exe --send-only exercise.example.net 11111
                                              # run on Windows as administrator
ls -l testdump.crash                          # run on styx, showing the dump file
vol.py -f testdump-net.raw imageinfo          # test the image integrity with Volatility
```

Answer to task 1.1

The following screenshot shows dumping to disk with WinPmem, in Windows Crashdump format. A first verification step is to check whether the file size of the dump file is roughly equal to the amount of RAM installed on the system the dump was taken from (with **dir** or **ls**). The commands on Windows must be run as administrator.

```
C:\Users\malbox\aq-tools>winpmem_1.4.exe -d testdump.crash
Driver Unloaded.
Loaded Driver C:\Users\malbox\AppData\Local\Temp\pme6B2.tmp.
Setting acquitision mode to 1
Will write a crash dump file
CR3: 0x0000185000
 2 memory ranges:
Start 0x00001000 - Length 0x0009E000
Start 0x00100000 - Length 0x1FEF0000

00% 0x00001000 .

00% 0x00100000 ................................................
09% 0x03300000 ................................................
19% 0x06500000 ................................................
29% 0x09700000 ................................................
```

```
39% 0x0C900000 ..............................................
49% 0x0FB00000 ..............................................
58% 0x12D00000 ..............................................
68% 0x15F00000 ..............................................
78% 0x19100000 ..............................................
88% 0x1C300000 ..............................................
97% 0x1F500000 ..........
Driver Unloaded.

C:\Users\malbox\aq-tools>dir testdump.crash
 Volume in drive C has no label.
 Volume Serial Number is 10AA-2D6C

 Directory of C:\Users\malbox\aq-tools>

08/14/2014  10:08 PM       536,408,064 testdump.crash
               1 File(s)    536,408,064 bytes
               0 Dir(s)     525,869,056 bytes free

C:\Users\malbox\aq-tools>
```

Example of a memory dump in Windows Crashdump format (-d) (over the network):

```
C:\Users\malbox\aq-tools>winpmem_1.4.exe -d - | ncat.exe --send-only exercise.example.net
11111


C:\Users\malbox\Desktop\winpmem-1.4>
```

The image can (and should) be verified with Volatility's **imageinfo**[50] command. The teacher should explain the fields in the output in detail (like AS Layer, etc.).

```
> vol.py -f testdump-net.raw imageinfo
Volatility Foundation Volatility Framework 2.3.1
Determining profile based on KDBG search...

          Suggested Profile(s) : Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/exercise/testdump-net.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82758be8
          Number of Processors : 1
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0x82759c00
```

[50] https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#imageinfo (last visited: 30. 9. 2014)

```
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2014-08-14 16:36:11 UTC+0000
     Image local date and time : 2014-08-14 18:36:11 +0200
```

Alternatively the image can be taken in Windows crashdump[48] format. The difference between the two formats as well as their advantages and disadvantages should be explained.

```
> vol.py -f testdump-net.crash imageinfo
Volatility Foundation Volatility Framework 2.3.1
Determining profile based on KDBG search...

          Suggested Profile(s) : Win7SP0x86, Win7SP1x86 (Instantiated with WinXPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : WindowsCrashDumpSpace32 (Unnamed AS)
                     AS Layer3 : FileAddressSpace (/home/exercise/testdump-net.crash)
                      PAE type : PAE
                           DTB : 0x185000L
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2014-08-14 16:04:20 UTC+0000
     Image local date and time : 2014-08-14 18:04:20 +0200
```

2.2.2 Task 1.2 Acquiring memory images from Linux

There are several ways of taking a memory image from a Linux systems. In addition to dumping the virtual machine memory (VMware, VirtualBox, KVM, etc.) or taking images through the Firewire interface, there are two ways of taking images from inside a running system through live acquisition tools:

1. Copying /dev/mem directly with userspace tools.
   The typical command could be as follows:

   dd if=/dev/mem of=test.dump bs=512 conv=noerror

   However, this works only if the **CONFIG_STRICT_DEVMEM** kernel option is not set.[51] One can look for it in **/boot/config-$(uname -r)**. If it is set, userspace reads from **/dev/mem** or **/dev/kmem** can't go beyond the first megabyte. If it is not set, LiME (see below) isn't needed and a dump from the kernel can be taken in the aforementioned way . Using LiME has two other advantages: First, it can be used on Android systems and second, it can dump in a format compatible with standard kernel debuggers.
   The first screenshot shows the output with the option switched to “on”, the second with the option switched to “off”. Root privileges are needed to access /dev/mem (as shown by the “#” command prompt).

```
> grep CONFIG_STRICT_DEVMEM /boot/config-2.6.32-34-server
CONFIG_STRICT_DEVMEM=y
# dd if=/dev/mem of=/dev/null bs=1M
```

[51] http://cateee.net/lkddb/web-lkddb/STRICT_DEVMEM.html (last visited: 30. 9. 2014)

```
dd: reading '/dev/mem': Operation not permitted
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.204055 s, 5.1 MB/s
```

```
> grep CONFIG_STRICT_DEVMEM /boot/config-3.1.10-1.16-desktop
# CONFIG_STRICT_DEVMEM is not set
# dd if=/dev/mem of=/dev/null bs=1M
dd: reading '/dev/mem': Bad address
991+0 records in
991+0 records out
1039138816 bytes (1.0 GB) copied, 1.84746 s, 562 MB/s
```

2. Using a special kernel module, here LiME.

   From the LiME website[52]: A Loadable Kernel Module (LKM) which allows for volatile memory acquisition from Linux and Linux-based devices, such as Android. This makes LiME unique as it is the first tool that allows for full memory captures on Android devices. It also minimizes its interaction between user and kernel space processes during acquisition, which allows it to produce memory captures that are more forensically sound than those of other tools designed for Linux memory acquisition.

   Some Linux distributions (like Ubuntu) have LiME as a package, which can be installed in the usual way. Others may have to build the LiME kernel module from source.

2.2.2.1 Task 1.2.1 Building the LiME kernel module

Answer to task 1.2.1:

Here: installing LiME on OpenSuSE:

- Download package:

  wget https://lime-forensics.googlecode.com/files/
  lime-forensics-1.1-r17.tar.gz

- Unpack:

  tar xvfz lime-forensics-1.1-r17.tar.gz

- Compile:
  This needs basic compile tools such as (g)cc and make as well as the Linux kernel header files matching the installed kernel. The version of the running kernel can be found with **uname -r**, architecture with **uname -p**
  cd src
  make
  Ignore the warnings. This should produce a Linux kernel module named **"lime-$(uname -r).ko"**

```
> cd src
> make
```

[52] https://github.com/504ensicsLabs/LiME (last visited: 30. 9. 2014)

```
make -C /lib/modules/3.4.6-2.10-default/build M=/home/exercise/src modules
make[1]: Entering directory `/usr/src/linux-3.4.6-2.10-obj/i386/default'
  CC [M]  /home/exercise/src/tcp.o
  CC [M]  /home/exercise/src/disk.o
  CC [M]  /home/exercise/src/main.o
/home/exercise/src/main.c: In function '__check_dio':
/home/exercise/src/main.c:56:1: warning: return from incompatible pointer type
[enabled by default]
  LD [M]  /home/exercise/src/lime.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/exercise/src/lime.mod.o
  LD [M]  /home/exercise/src/lime.ko
make[1]: Leaving directory `/usr/src/linux-3.4.6-2.10-obj/i386/default'
strip --strip-unneeded lime.ko
mv lime.ko lime-3.4.6-2.10-default.ko
make tidy
make[1]: Entering directory `/home/exercise/src'
rm -f *.o *.mod.c Module.symvers Module.markers modules.order \.*.o.cmd \.*.ko.cmd
\.*.o.d
rm -rf \.tmp_versions
make[1]: Leaving directory `/home/exercise/src'
> uname -r
3.4.6-2.10-default
> ls -l
insgesamt 36
-rw-r--r-- 1 exercise users 2117 19. Mär 2013  disk.c
-rw-r--r-- 1 exercise users 6908 11. Aug 23:53 lime-3.4.6-2.10-default.ko
-rw-r--r-- 1 exercise users 1861 19. Mär 2013  lime.h
-rw-r--r-- 1 exercise users 5028 19. Mär 2013  main.c
-rw-r--r-- 1 exercise users 1248 19. Mär 2013  Makefile
-rw-r--r-- 1 exercise users 1723 19. Mär 2013  Makefile.sample
-rw-r--r-- 1 exercise users 3160 19. Mär 2013  tcp.c
```

As the source package is already installed on the Opsu appliance, only the following commands are needed for the exercise:

```
cd src
make
uname -r
ls -l
```

2.2.2.2 Task 1.2.2 Taking the image from a Linux system

The LiME kernel module needs at least two parameters:

1. The location where the dump should go to, this is supplied with the **path** parameter **path** is either a full pathname of the dump file or a TCP port number on the local system:

path=/tmp/dump-filename or **path=tcp:23456**. Of course, there must be a program listening on the port to forward the data to its final destination.

2. The format of the dump, supplied with the **format** parameter **(format** can be **raw**, **padded** or **lime**) . As Volatility directly supports the **lime** address space, this should be used when dumping for use with Volatility. For older versions of Volatility or other tools that do not support lime address space, **raw** is likely the best choice.

Answer to task 1.2.2:

The actual dump is done by installing the LiME kernel module with the appropriate parameters (as root).

insmod lime.ko "path=/media/usb-stick/memdump.lime format=raw"

```
# insmod lime-3.4.6-2.10-default.ko "path=/tmp/testdump.lime format=lime"
# ls -l /tmp/testdump.lime
-r--r--r-- 1 root root 536345664 12. Aug 00:05 /tmp/testdump.lime
#
```

When doing several dumps, ensure to unload the **lime** module before the next dump with **rmmod lime** (unlike the previous **insmod** command, the version number of the kernel module is not needed here).

Commands for task 1.2.2 (the **insmod** command must be executed as root).

```
insmod lime-3.4.6-2.10-default.ko "path=/tmp/testdump.lime format=lime"
ls -l /tmp/testdump.lime
```

As a quick check, the dump file should almost be the same size as the amount of RAM installed on the system the dump was taken from. (In the example: 512 MB).

```
# insmod lime-3.4.6-2.10-default.ko "path=/tmp/testdump.lime format=lime"
# ls -l /tmp/testdump.lime
-r--r--r-- 1 root root 536345664 12. Aug 00:05 /tmp/testdump.lime
#
```

As another check, one can look into the dump's header. This can be useful if the format has been forgotten. As can be seen, LiME dumps have a clealy visible signature (EMiL).

```
> hexdump -C /tmp/testdump.raw | head -3
00000000  36 63 62 33 65 39 63 63  30 65 66 38 33 37 65 39  |6cb3e9cc0ef837e9|
00000010  65 31 36 39 65 39 63 32  64 33 38 63 36 30 61 37  |e169e9c2d38c60a7|
00000020  00 34 39 63 36 62 32 66  35 30 65 62 66 35 32 36  |.49c6b2f50ebf526|

> hexdump -C /tmp/testdump.lime | head -3
00000000  45 4d 69 4c 01 00 00 00  00 00 01 00 00 00 00 00  |EMiL............|
00000010  ff fb 09 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  36 63 62 33 65 39 63 63  30 65 66 38 33 37 65 39  |6cb3e9cc0ef837e9|

> hexdump -C /tmp/testdump.padded | head -3
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

```
00010000  36 63 62 33 65 39 63 63  30 65 66 38 33 37 65 39  |6cb3e9cc0ef837e9|
```

2.2.2.3 Task 1.2.3 Building a Volatility profile

Profiles must match exactly the:

- OS Type (CentOS, OpenSuSE, Ubuntu, …).
- Architecture (x86, x64, ARM, …).
- OS Version (uname -r).

If a matching profile is not available, a matching profile can be built:

Answer to task 1.2.3:

Building a profile on/for OpenSuSE 12.2:

1. Copy the directory **/usr/src/volatility-tools/linux** and its contents to the linux system that requires a profile.
2. Install the Linux kernel header of the version of the OS running, i.e. if 3.4.6-2.10-default is running on the system (**uname -r**), one shoud install kernel-source-3.4.6-2.10.1.noarch (Fedora, OpenSuSE). Furthermore, you need the **dwarfdump** program supplied with the **dwarfdump** package on Debian systems or **libdwarf-tools** (Fedora, OpenSuSE). The distribution specific tools like yum search (Fedora), zypper search (OpenSuSE), or apt-get search (Debian) should be used to search for an appropriate package.
3. In the **linux** directory, run **make**. This builds **module.dwarf.**

```
> make
make -C //lib/modules/3.4.6-2.10-default/build CONFIG_DEBUG_INFO=y
M=/home/exercise/volatility-tools/linux modules
make[1]: Entering directory `/usr/src/linux-3.4.6-2.10-obj/i386/default'
  CC [M]  /home/exercise/volatility-tools/linux/module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/exercise/volatility-tools/linux/module.mod.o
  LD [M]  /home/exercise/volatility-tools/linux/module.ko
make[1]: Leaving directory `/usr/src/linux-3.4.6-2.10-obj/i386/default'
dwarfdump -di module.ko > module.dwarf
make -C //lib/modules/3.4.6-2.10-default/build M=/home/exercise/volatility-tools/linux
clean
make[1]: Entering directory `/usr/src/linux-3.4.6-2.10-obj/i386/default'
  CLEAN   /home/exercise/volatility-tools/linux/.tmp_versions
  CLEAN   /home/exercise/volatility-tools/linux/Module.symvers
make[1]: Leaving directory `/usr/src/linux-3.4.6-2.10-obj/i386/default'
```

4. You need **System.map** file of the running kernel, which is typically in directory **/boot** like **/boot/System.map-3.4.6-2.10-default** for example.
5. Copy both files (**module.dwarf** and **System.map**) into a zip file (this step can be done on another system if the system you compiled on doesn't have the zip command installed). Give the file a name:
   zip OpenSuSE12.2-x86.zip module.dwarf /boot/System.map-3.4.6-2.10-default.

6. Copy the Zip file into the users profile folder or into the Volatility profiles folder **plugins/overlays/linux** (on Debian/Ubuntu systems, this will be **/usr/lib/python2.7/dist-packages/volatility/**). This will require root privileges and the folder may be overwritten by updates or re-installs of Volatility (better keep a backup of the self-made profiles somewhere else).
7. Verify that Volatility finds the profile:

```
> vol.py --info | grep -i suse
Volatility Foundation Volatility Framework 2.3.1
LinuxOpenSuSE12_2-x86x86                                    - A Profile for
Linux OpenSuSE12.2-x86 x86
```

The tools and source package are already installed on the OpSu appliance.

Commands used:

```
cd volatility-tools/linux
make
zip OpenSuSE12.2-x86.zip module.dwarf /boot/System.map-3.4.6-2.10-default
```

Caveat: If a student copy of Volatility can't find its profile folder, it may be because of a bug where **--plugins** has to be the first option supplied, i. e. **vol.py --plugins --info**, not **vol.py --info --plugins**

To see if your profile works correctly, try commands with the profile, like **linux_cpuinfo** or **linux_banner**. The example below uses the previously created **testdump.lime** file.

```
> vol.py -f testdump.lime --profile LinuxOpenSuSE12_2-x86x86 linux_banner
Volatility Foundation Volatility Framework 2.3.1
Linux version 3.4.6-2.10-default (geeko@buildhost) (gcc version 4.7.1 20120723 [gcc-4_7-branch
revision 189773] (SUSE Linux) ) #1 SMP Thu Jul 26 09:36:26 UTC 2012 (641c197)
> vol.py -f testdump.lime --profile LinuxOpenSuSE12_2-x86x86 linux_cpuinfo
Volatility Foundation Volatility Framework 2.3.1
Processor    Vendor           Model
------------ ---------------- -----
0            GenuineIntel     Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
```

The same dump with a slightly different profile (see below):

```
> vol.py -f testdump.lime --profile LinuxOpenSuSE122x86 linux_cpuinfo
Volatility Foundation Volatility Framework 2.3.1
ERROR   : volatility.addrspace: Invalid profile LinuxOpenSuSE122x86 selected
> vol.py -f testdump.lime --profile LinuxOpenSuSE122x86 linux_banner
Volatility Foundation Volatility Framework 2.3.1
ERROR   : volatility.addrspace: Invalid profile LinuxOpenSuSE122x86 selected
```

So, what's going wrong, both profiles match the OS version (OpenSuSE 12.2) the architecture (x86) and kernel version (3.4.6-10)?

The filename of a profile often does not tell the exact kernel version it was taken from. With **unzip**, one can look into the profile, specifically at the filename of the **System.map** file to get more details.

```
> unzip -l OpenSuSE12.2-x86.zip
Archive:  OpenSuSE12.2-x86.zip
```

```
  Length      Date    Time    Name
---------  ---------- -----   ----
  1610967  2014-08-11 23:08   module.dwarf
  1863231  2012-08-04 10:30   boot/System.map-3.4.6-2.10-default
---------                     -------
  3474198                     2 files
> unzip -l OpenSUSE122.zip
Archive:  OpenSUSE122.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
  1630818  2014-02-01 15:55   volatility/tools/linux/module.dwarf
  1969517  2012-08-04 10:15   boot/System.map-3.4.6-2.10-desktop
---------                     -------
  3600335                     2 files
```

As can be seen, even a very minor difference (3.4.6-2.10-default vs. 3.4.6-2.10-desktop) can render a profile unusable. The versions must match **exactly**!

Ubuntu/Debian distributions have the **volatility-profiles** package which contains profiles for many versions of Debian or Ubuntu. However, not all in-between patch levels of the kernels are in this package.

2.3 Task 2: Basic analysis of Windows images

In the introductory part of this task, the teacher should introduce Volatility commands for process analysis, how they work, and their output[53].

- **pslist, pstree, psscan, threads, thrdscan, psxview**

Aside from the process listings, there are several more commands that should be introduced, as they give further information about a process and some of them will be needed later on in the exercise. It is assumed that the students have a basic knowledge about what environment variables, Windows privileges, and SIDs are but if there a questions from the students, the teacher should be prepared to handle them.

- **dlllist, handles, envars, privs, getsids,**

Finally, the **console** and **cmdscan** Volatility commands should be introduced.

2.3.1 Task 2.1: Windows process analysis

This task will use Windows memory image, **windows-zeus.vmem** - an image of an x86 Windows XP system infected with the Zeus banking trojan. The **WinXPSP2x86** profile should be used. As a second image, the one taken from task 1.1 can be used.

Commands used for the task:

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem pslist
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem dlllist -p 856
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem envars -p 856
```

[53] https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#processes-and-dlls (last visited: 30. 9. 2014)

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem privs -p 856
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem getsids -p 856
```

Questions:

In the image, find the process with PID 856.

1. What program is running there, and what are its command line arguments?
2. What is its parent process?
3. What are the environment variables of this process?
4. What are the enabled privileges?
5. Which SIDs is this process running with? Is this normal?

Answer to question 1: What program is running there, and what are its command line arguments?

The program is called svchost.exe. It is launched with the command line **C:\WINDOWS\system32\svchost -k DcomLaunch.**

Here's the sample output from the **pslist** command.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem pslist
Volatility Foundation Volatility Framework 2.3.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start  Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------ ------
0x810b1660 System                    4      0     58      379 ------      0
0xff2ab020 smss.exe                544      4      3       21 ------      0 2010-08-11
06:06:21 UTC+0000
0xff1ecda0 csrss.exe               608    544     10      410      0      0 2010-08-11
06:06:23 UTC+0000
0xff1ec978 winlogon.exe            632    544     24      536      0      0 2010-08-11
06:06:23 UTC+0000
0xff247020 services.exe            676    632     16      288      0      0 2010-08-11
06:06:24 UTC+0000
0xff255020 lsass.exe               688    632     21      405      0      0 2010-08-11
06:06:24 UTC+0000
0xff218230 vmacthlp.exe            844    676      1       37      0      0 2010-08-11
06:06:24 UTC+0000
0x80ff88d8 svchost.exe             856    676     29      336      0      0 2010-08-11
06:06:24 UTC+0000
0xff217560 svchost.exe             936    676     11      288      0      0 2010-08-11
06:06:24 UTC+0000
… [truncated] …
0xff224020 cmd.exe                 124   1668      0 --------      0      0 2010-08-15
19:17:55 UTC+0000   2010-08-15 19:17:56 UTC+0000
```

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem dlllist -p 856
Volatility Foundation Volatility Framework 2.3.1
************************************************************************
svchost.exe pid:    856
Command line : C:\WINDOWS\system32\svchost -k DcomLaunch
```

```
Service Pack 2
Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x01000000     0x6000     0xffff C:\WINDOWS\system32\svchost.exe
0x7c900000    0xb0000     0xffff C:\WINDOWS\system32\ntdll.dll
… [truncated] …
```

Answer to question 2: What is its parent process?

It's parent process has the PID 676, services.exe. See the output above.

Answer to question 3: What are the environment variables of this process?

It's environment variables are given in the listing below.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem envars -p 856
Volatility Foundation Volatility Framework 2.3.1
Pid  Process      Block      Variable                  Value
---- -----------  ---------- ------------------------  -----
 856 svchost.exe  0x00010000 ALLUSERSPROFILE           C:\Documents and Settings\ All Users
 856 svchost.exe  0x00010000 CommonProgramFiles        C:\Program Files\Common Files
 856 svchost.exe  0x00010000 COMPUTERNAME              BILLY-DB5B96DD3
 856 svchost.exe  0x00010000 ComSpec                   C:\WINDOWS\system32\cmd.exe
 856 svchost.exe  0x00010000 FP_NO_HOST_CHECK          NO
 856 svchost.exe  0x00010000 NUMBER_OF_PROCESSORS      1
 856 svchost.exe  0x00010000 OS                        Windows_NT
 856 svchost.exe  0x00010000 Path                      C:\WINDOWS\system32;
C:\WINDOWS;C:\WINDOWS\System32\Wbem
 856 svchost.exe  0x00010000
PATHEXT                   .COM;.EXE;.BAT;.CMD;.VBS; .VBE;.JS;.JSE;.WSF;.WSH
 856 svchost.exe  0x00010000 PROCESSOR_ARCHITECTURE    x86
 856 svchost.exe  0x00010000 PROCESSOR_IDENTIFIER      x86 Family 6 Model 23 Stepping 10,
GenuineIntel
 856 svchost.exe  0x00010000 PROCESSOR_LEVEL           6
 856 svchost.exe  0x00010000 PROCESSOR_REVISION        170a
 856 svchost.exe  0x00010000 ProgramFiles              C:\Program Files
 856 svchost.exe  0x00010000 SystemDrive               C:
 856 svchost.exe  0x00010000 SystemRoot                C:\WINDOWS
 856 svchost.exe  0x00010000 TEMP                      C:\WINDOWS\TEMP
 856 svchost.exe  0x00010000 TMP                       C:\WINDOWS\TEMP
 856 svchost.exe  0x00010000 USERPROFILE               C:\WINDOWS\system32
\config\systemprofile
 856 svchost.exe  0x00010000 windir                    C:\WINDOWS
```

Answer to question 4: What are the enabled privileges?

A listing of the privileges is given below.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem privs -p 856
```

```
Volatility Foundation Volatility Framework 2.3.1
Pid   Process      Value  Privilege                         Attributes    Description
----  ------------ ------ --------------------------------- ------------  ---------
 856  svchost.exe       7 SeTcbPrivilege                    Present,Enabled,Default  Act as part
of the operating system
 856  svchost.exe       2 SeCreateTokenPrivilege            Present                  Create a
token object
 856  svchost.exe       9 SeTakeOwnershipPrivilege          Present                  Take
ownership of files/objects
 856  svchost.exe      15 SeCreatePagefilePrivilege         Present,Enabled,Default  Create a
pagefile
 856  svchost.exe       4 SeLockMemoryPrivilege             Present,Enabled,Default  Lock pages
in memory
 856  svchost.exe       3 SeAssignPrimaryTokenPrivilege     Present                  Replace a
process-level token
 856  svchost.exe       5 SeIncreaseQuotaPrivilege          Present                  Increase
quotas
 856  svchost.exe      14 SeIncreaseBasePriorityPrivilege   Present,Enabled,Default  Increase
scheduling priority
 856  svchost.exe      16 SeCreatePermanentPrivilege        Present,Enabled,Default  Create
permanent shared objects
 856  svchost.exe      20 SeDebugPrivilege                  Present,Enabled,Default  Debug
programs
 856  svchost.exe      21 SeAuditPrivilege                  Present,Enabled,Default  Generate
security audits
 856  svchost.exe       8 SeSecurityPrivilege               Present                  Manage
auditing and security log
 856  svchost.exe      22 SeSystemEnvironmentPrivilege      Present                  Edit
firmware environment values
 856  svchost.exe      23 SeChangeNotifyPrivilege           Present,Enabled,Default  Receive
notifications of changes to files or directories
 856  svchost.exe      17 SeBackupPrivilege                 Present                  Backup
files and directories
 856  svchost.exe      18 SeRestorePrivilege                Present                  Restore
files and directories
 856  svchost.exe      19 SeShutdownPrivilege               Present                  Shut down
the system
 856  svchost.exe      10 SeLoadDriverPrivilege             Present                  Load and
unload device drivers
 856  svchost.exe      13 SeProfileSingleProcessPrivilege   Present,Enabled,Default  Profile a
single process
 856  svchost.exe      12 SeSystemtimePrivilege             Present                  Change the
system time
 856  svchost.exe      25 SeUndockPrivilege                 Present                  Remove
computer from docking station
 856  svchost.exe      28 SeManageVolumePrivilege           Present                  Manage the
files on a volume
 856  svchost.exe      29 SeImpersonatePrivilege            Present,Enabled,Default  Impersonate
a client after authentication
 856  svchost.exe      30 SeCreateGlobalPrivilege           Present,Enabled,Default  Create
global objects
```

Answer to question 5: Which SIDs is this process running with? Is this common?

The SIDs are:

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem getsids -p 856
Volatility Foundation Volatility Framework 2.3.1
svchost.exe (856): S-1-5-18 (Local System)
svchost.exe (856): S-1-5-32-544 (Administrators)
svchost.exe (856): S-1-1-0 (Everyone)
svchost.exe (856): S-1-5-11 (Authenticated Users)
```

This is not common, the other **svchost.exe** processes do not run with administrator SIDs. Compare with the PIDs 936, 1028, 1088 or 1148.

2.3.2 Task 2.2: Network connection analysis

The teacher should explain that there are differences in the network stack of the NT 5.1 kernel (XP, Server 2003) and the 6.x (Vista and newer) series of kernels. Thus different plugins are used.

- **connections** and **connscan** for open connections.[54]
- **sockets** and **sockscan** for open sockets.[54]

Neither of them works on IPv6 sockets.

- **netscan**[54] for sockets and connections, IPv4 and IPv6, but only on Vista and newer.

Commands used:

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem connections
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem connscan
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem sockets
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem sockscan
```

Questions:

1. Do an analysis of the network communications in the image.
2. Try to estimate which of the communications could be malicious.

Answer to question 1:

The connections command shows nothing:

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem connections
Volatility Foundation Volatility Framework 2.3.1
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
```

However, **connscan** gives a connection and an IP address.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem connscan
Volatility Foundation Volatility Framework 2.3.1
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
```

[54] https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#networking (last visited: 30. 9. 2014)

```
0x02214988 172.16.176.143:1054       193.104.41.75:80          856
0x06015ab0 0.0.0.0:1056              193.104.41.75:80          856
```

Answer to question 2:

And that IP address looks suspicious (what does that system want in Transnistria?). Even if that isn't suspicious, svchost.exe doesn't open connections by itself.

```
> host 193.104.41.75
Host 75.41.104.193.in-addr.arpa. not found: 3(NXDOMAIN)
> whois 193.104.41.75
…
inetnum:        193.104.41.0 - 193.104.41.255
netname:        VVPN-NET
…
person:         Evgen Sergeevich Voronov
address:        25 October street, 118-15
address:        Tiraspol, Transdnistria
```

A further scan with sockets yields no further evidence. Just normal sockets and an open port 29220 hold by the process with pid 856 (but we have encountered this process already).

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem sockets
Volatility Foundation Volatility Framework 2.3.1
Offset(V)       PID   Port  Proto Protocol        Address         Create Time
---------- -------- ------ ------ --------------- --------------- -----------
0x80fd1008        4      0     47 GRE             0.0.0.0         2010-08-11 06:08:00 UTC+0000
0xff258008      688    500     17 UDP             0.0.0.0         2010-08-11 06:06:35 UTC+0000
0xff367008        4    445      6 TCP             0.0.0.0         2010-08-11 06:06:17 UTC+0000
0x80ffc128      936    135      6 TCP             0.0.0.0         2010-08-11 06:06:24 UTC+0000
0xff37cd28     1028   1058      6 TCP             0.0.0.0         2010-08-15 19:17:56 UTC+0000
0xff20c478      856  29220      6 TCP             0.0.0.0         2010-08-15 19:17:27 UTC+0000
0xff225b70      688      0    255 Reserved        0.0.0.0         2010-08-11 06:06:35 UTC+0000
0xff254008     1028    123     17 UDP             127.0.0.1       2010-08-15 19:17:56 UTC+0000
0x80fce930     1088   1025     17 UDP             0.0.0.0         2010-08-11 06:06:38 UTC+0000
0xff127d28      216   1026      6 TCP             127.0.0.1       2010-08-11 06:06:39 UTC+0000
0xff206a20     1148   1900     17 UDP             127.0.0.1       2010-08-15 19:17:56 UTC+0000
0xff1b8250      688   4500     17 UDP             0.0.0.0         2010-08-11 06:06:35 UTC+0000
0xff382e98        4   1033      6 TCP             0.0.0.0         2010-08-11 06:08:00 UTC+0000
0x80fbdc40        4    445     17 UDP             0.0.0.0         2010-08-11 06:06:17 UTC+0000
```

2.3.3 Task 2.3: Registry analysis

The teacher should introduce the concept of registry hives in memory backed up by files on disk, and the difference between stable and volatile registry keys[16]. The following commands should be explained: **hivelist, hivedump, printkey**[55]. As there are hundreds of keys that can be used in the

[55] https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#registry (last visited: 30. 9. 2014)

forensic analysis of a Windows system, the teacher should refer to the **Autoruns** and **RegRipper** tools for further reading.

The Local Security Authority (LSA) is the key process for Windows Security[16]. With students versed in Windows operating system concepts, this process would need no explanation. However, there might be students that have questions here. The commands **lsadump** and **hashdump**[55] deal with locally stored credentials in the registry, maintained by the LSA.

Further, the **userassist** and **getservicesids**[55] commands should be introduced.

Commands used in the task:

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem hivelist

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem userassist

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -K "Microsoft\Windows
NT\CurrentVersion\Winlogon"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -K "Microsoft\Windows
NT\CurrentVersion\Winlogon"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem hashdump -y 0xe101b008 -s 0xe1544008
```

Questions to task 2.3:

1. Find out if there is a malicious UserAssist key in the registry.
2. What is the value of the “Microsoft\Windows NT\CurrentVersion\Winlogon” key? It is located in the “software” hive.
3. Analyse the “Windows Internet Connection Firewall” (ICF) settings. The keys are locatd under the “ControlSet001\Services\SharedAccess” key of the “System” hive.
4. What user accounts exist on the system?

Answers to task 2.3:

First a listing of the registry hives because the offsets are needed in the following commands:

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem hivelist

Volatility Foundation Volatility Framework 2.3.1

Virtual    Physical   Name

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

0xe1c49008 0x036dc008 \Device\HarddiskVolume1\Documents and Settings\ LocalService\Local
Settings\Application Data\Microsoft\Windows\UsrClass.dat

0xe1c41b60 0x04010b60 \Device\HarddiskVolume1\Documents and Settings\ LocalService\NTUSER.DAT

0xe1a39638 0x021eb638 \Device\HarddiskVolume1\Documents and Settings\ NetworkService\Local
Settings\Application Data\Microsoft\Windows\UsrClass.dat

0xe1a33008 0x01f98008 \Device\HarddiskVolume1\Documents and Settings\
NetworkService\NTUSER.DAT

0xe153ab60 0x06b7db60 \Device\HarddiskVolume1\WINDOWS\system32\config\software
```

```
0xe1542008 0x06c48008 \Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1537b60 0x06ae4b60 \SystemRoot\System32\Config\SECURITY
0xe1544008 0x06c4b008 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe13ae580 0x01bbd580 [no name]
0xe101b008 0x01867008 \Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe1008978 0x01824978 [no name]
0xe1e158c0 0x009728c0 \Device\HarddiskVolume1\Documents and Settings\ Administrator\Local
Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1da4008 0x00f6e008 \Device\HarddiskVolume1\Documents and Settings\ Administrator\NTUSER.DAT
```

Answer to question 1: Is there a malicious UserAssist Key?

Even without the telling name, the Value UEME_Runpath stands out and would justify further examination.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem userassist
Volatility Foundation Volatility Framework 2.3.1
----------------------------
Registry: \Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
Key name: Count
Last updated: 2010-08-15 19:17:23 UTC+0000
Subkeys:
Values:
REG_BINARY    UEME_CTLSESSION :
0x00000000  d7 c8 59 0e 02 00 00 00                           ..Y.....
… [truncated] …
REG_BINARY    UEME_RUNPATH:C:\Documents and Settings\Administrator
\Desktop\ZeuS_binary_5767b2c6d84d87a47d12da03f4f376ad.exe :
ID:             2
Count:          1
Last updated:   2010-08-15 19:17:23 UTC+0000
0x00000000  02 00 00 00 06 00 00 00 60 35 58 7d ae 3c cb 01    ........`5X}.<..
… [truncated] …
```

Answer to question 2: What is the value of the "Winlogon" key?

We need the offset of the "software" hive (**0xe153ab60**) from above. Interesting is the "UserInit" value, as it is executed at logon of a user.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe153ab60 -K
"Microsoft\Windows NT\CurrentVersion\Winlogon"
Volatility Foundation Volatility Framework 2.3.1
Legend: (S) = Stable   (V) = Volatile
----------------------------
Registry: User Specified
Key name: Winlogon (S)
Last updated: 2010-08-15 19:17:23 UTC+0000
Subkeys:
  (S) GPExtensions
```

```
  (S) Notify
  (S) SpecialAccounts
  (V) Credentials
Values:
REG_DWORD     AutoRestartShell : (S) 1
REG_SZ        DefaultDomainName : (S) BILLY-DB5B96DD3
REG_SZ        DefaultUserName : (S) Administrator
REG_SZ        LegalNoticeCaption : (S)
REG_SZ        LegalNoticeText : (S)
REG_SZ        PowerdownAfterShutdown : (S) 0
REG_SZ        ReportBootOk    : (S) 1
REG_SZ        Shell           : (S) Explorer.exe
REG_SZ        ShutdownWithoutLogon : (S) 0
REG_SZ        System          : (S)
REG_SZ        Userinit        : (S) C:\WINDOWS\system32\userinit.exe,
C:\WINDOWS\system32\sdra64.exe,
REG_SZ        VmApplet        : (S) rundll32 shell32,Control_RunDLL "sysdm.cpl"
… [truncated] …
```

Answer to question 3: Analyse the ICF settings

For the firewall settings, the student has to take the offset of the hive (**0xe101b008** see above) and then to iterate through several keys:

"ControlSet001\Services\SharedAccess" "ControlSet001\Services\SharedAccess\Parameters" "ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy" and finally

"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile" before the value of **EnableFirewall** is seen. It is disabled (0), but should be enabled (1) by default.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy \StandardProfile"
Volatility Foundation Volatility Framework 2.3.1
Legend: (S) = Stable   (V) = Volatile
----------------------------
Registry: User Specified
Key name: StandardProfile (S)
Last updated: 2010-08-15 19:17:24 UTC+0000
Subkeys:
  (S) AuthorizedApplications
Values:
REG_DWORD     EnableFirewall  : (S) 0
```

Answer to question 4: What user accounts exist on the system?

The user account list resides in the SAM hive of the Windows registry (at least for the local system accounts, domain accounts are stored in the active directory of the domain controller). Thus, the

offsets for the system (-y) and SAM (-s) hive are needed from the above hivelist. Just the standard accounts here, not even an account for an unprivileged user.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem hashdump -y 0xe101b008 -s 0xe1544008
Volatility Foundation Volatility Framework 2.3.1
Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eaee8fb117ad06bdd830b7586c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:4e857c004024e53cd538de64dedac36b:842b4013c45a3b8fec76ca54e5910581:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:8f57385a61425fc7874c3268aa249ea1:::
```

2.3.4 Task 2.4: File and other analysis

For the file analysis the **mbrparser** and **mftparser** plugins should be introduced[56]. However, they are not needed for the task. What are needed are the **consoles** and **cmdscan** plugins as well as the **strings** plugin.

Commands used in this task:

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem handles -t File | grep sdra
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem pstree
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem handles -p632,676,856 -t File,Mutant | grep -i avira
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem consoles
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem cmdscan
```

Questions to task 2.4:

1. Building on the task before, find the “sdra64.exe” file. What process has an open file handle to it?
2. Examining the suspicious processes from the subtasks before, what stands out in the list of objects the process has opened?
3. Can you find out which commands were typed in the terminated cmd.exe process?

Answer to question 1: What process has an open handle to the sdra64.exe file?

With the handles plugin, we can search for processes that have this file open. Here it is the PID 632 (winlogon.exe).

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem handles -t File | grep sdra
Volatility Foundation Volatility Framework 2.3.1
0xff12bb40    632      0x644    0x120089 File
\Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe
>
```

What makes this interesting is that this process is the parent of services.exe, which in turn is the parent of the suspicious svchost.exe (see pstree listing).

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem pstree
Volatility Foundation Volatility Framework 2.3.1
```

[56] https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#file-system (last visited: 30. 9. 2014)

```
Name                                                 Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x810b1660:System                                     4      0     58    379 1970-01-01
00:00:00 UTC+0000
. 0xff2ab020:smss.exe                                544      4      3     21 2010-08-11
06:06:21 UTC+0000
.. 0xff1ec978:winlogon.exe                           632    544     24    536 2010-08-11
06:06:23 UTC+0000
... 0xff255020:lsass.exe                             688    632     21    405 2010-08-11
06:06:24 UTC+0000
... 0xff247020:services.exe                          676    632     16    288 2010-08-11
06:06:24 UTC+0000
.... 0xff1b8b28:vmtoolsd.exe                        1668    676      5    225 2010-08-11
06:06:35 UTC+0000
..... 0xff224020:cmd.exe                             124   1668      0 ------ 2010-08-15
19:17:55 UTC+0000
.... 0x80ff88d8:svchost.exe                          856    676     29    336 2010-08-11
06:06:24 UTC+0000
.... 0xff1d7da0:spoolsv.exe                         1432    676     14    145 2010-08-11
06:06:26 UTC+0000
… [truncated] …
```

Answer to question 2: What stands out in the list of objects?

Looking again with the handles plugin at these three processes, we notice the two very similarly named "named pipes": "_AVIRA_2109" and "_AVIRA_2108". Note also the correspondingly named Mutex objects.

(The unfiltered output is quite long, so for brevity, only the filtered part is shown here).

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem handles -p632,676,856 -t File,Mutant |
grep -i avira
Volatility Foundation Volatility Framework 2.3.1
0x80f5cd78    632      0x898    0x12019f File             \Device\NamedPipe\_AVIRA_2109
0xff1e7dc0     632      0x8bc    0x1f0001 Mutant           _AVIRA_2109
0xff1398c0     856      0x434    0x12019f File             \Device\NamedPipe\_AVIRA_2108
0xff27b7e8     856      0x43c    0x1f0001 Mutant           _AVIRA_2108
```

Answer to question 3: Can you find out which commands were typed in the terminated cmd.exe process?

The commands entered at the cmd.exe process could not be reconstructed, as per output of Volatility's **consoles** and **cmdscan** plugins.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem consoles
Volatility Foundation Volatility Framework 2.3.1
**************************************************
ConsoleProcess: csrss.exe Pid: 608
Console: 0x4e23b0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
Title: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
AttachedProcess: TPAutoConnect.e Pid: 1084 Handle: 0x448
```

```
----
CommandHistory: 0xf786f8 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x448
----
Screen 0x4e2ab0 X:80 Y:25
Dump:
TPAutoConnect User Agent, Copyright (c) 1999-2009 ThinPrint AG, 7.17.512.1
************************************************
ConsoleProcess: csrss.exe Pid: 608
Console: 0xf78958 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: ??ystemRoot%\system32\cmd.exe
Title:
```

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem cmdscan
Volatility Foundation Volatility Framework 2.3.1
************************************************
CommandProcess: csrss.exe Pid: 608
CommandHistory: 0xf786f8 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x448
```

2.4 Task 3: Basic Linux memory analysis

The introductory part will start with the basic Linux commands [57], notably **linux_cpuinfo** and **linux_dmesg**.

A large part will take an introduction to Linux kernel memory allocation and some of its commands[57]: **linux_iomem, linux_slabinfo** and **linux_vma_cache**. The teacher should also explain the Linux kernel allocators SLAB[58], SLUB[59] and SLOB[60] and that only the first one is (fully) supported by Volatility. However, all systems in the exercise use the SLAB allocator.

2.4.1 Task 3.1: Linux process analysis

The teacher will first introduce the commands used by Volatility to do basic Linux process listings[61]: **linux_pslist, linux_psaux, linux_pstree, linux_pslist_cache, linux_pidhashtable** and **linux_psxview**.

[57] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#system-information (last visited: 30. 9. 2014)
[58] https://en.wikipedia.org/wiki/SLAB (last visited: 30. 9. 2014)
[59] https://en.wikipedia.org/wiki/SLUB (last visited: 30. 9. 2014)
[60] https://en.wikipedia.org/wiki/SLOB (last visited: 30. 9. 2014)
[61] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#processes (last visited: 30. 9. 2014)

Besides these, the **linux_lsof** should be introduced here, as it is used in the tasks below. It works similar to the **handles** command covered in the Windows section.

For this task, we will be using the kbeast image(s):

- **Ubuntu-10.04.3-i386-LiveCD-clean.mem**
- **Ubuntu-10.04.3-i386-LiveCD-kbeast.mem**

The profile is: **LinuxUbuntu10_04x86**. This is a custom profile supplied together with the Linux images in the same folder.

Commands used in this task

```
vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux cpuinfo

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_dmesg

vol.py --plugins=../vol-profiles --profile=LinuxOpenSuSE12_2-x86x86 -f opsu.lime linux_pstree

vol.py --plugins=../vol-profiles --profile=LinuxOpenSuSE12_2-x86x86 -f opsu.lime linux_bash
```

Questions to task 3.1:

1. What type of processor and how many of them are used in the system the kbeast images were taken from?
2. The “attacker” has connected a device to the system. Which one?
3. Take the image from task 1.2 and try to find your own forensic session processes.

Answer to question 1: What type of processor and how many of them are used in the system the kbeast images were taken from?

One processor. For details, see the **linux_cpuinfo** output:

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_cpuinfo

Volatility Foundation Volatility Framework 2.3.1

Processor    Vendor           Model

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

0            GenuineIntel     Intel(R) Core(TM) i7-2860QM CPU @ 2.50GHz
```

Answer to question 2: The “attacker” has connected a device to the system. Which one?

An 8GB “Kanguru FlashBlu” USB stick/drive. See **linux_dmesg** output:

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_dmesg

Volatility Foundation Volatility Framework 2.3.1

[2314885531810281020.2314885531] ] Initializing cgroup subsys cpuset

<6>[    0.000000] Initializing cgroup subsys cpu

… [truncated] ...

<6>[ 5404.140360] usb 1-2: new full speed USB device using ohci_hcd and address 5

<6>[ 5404.402409] usb 1-2: configuration #1 chosen from 1 choice

<6>[ 5404.923893] Initializing USB Mass Storage driver...

<6>[ 5404.925165] scsi3 : SCSI emulation for USB Mass Storage devices

<6>[ 5404.925515] usbcore: registered new interface driver usb-storage
```

```
<6>[ 5404.925517] USB Mass Storage support registered.
<7>[ 5404.930674] usb-storage: device found at 5
<7>[ 5404.930675] usb-storage: waiting for device to settle before scanning
<7>[ 5409.939032] usb-storage: device scan complete
<5>[ 5409.947870] scsi 3:0:0:0: Direct-Access     Kanguru  FlashBlu         PMAP PQ: 0 ANSI: 0
CCS
<5>[ 5409.952639] sd 3:0:0:0: Attached scsi generic sg2 type 0
<5>[ 5409.972223] sd 3:0:0:0: [sdb] 15646720 512-byte logical blocks: (8.01 GB/7.46 GiB)
<5>[ 5409.982096] sd 3:0:0:0: [sdb] Write Protect is off
<7>[ 5409.982098] sd 3:0:0:0: [sdb] Mode Sense: 23 00 00 00
<3>[ 5409.982099] sd 3:0:0:0: [sdb] Assuming drive cache: write through
<3>[ 5410.028186] sd 3:0:0:0: [sdb] Assuming drive cache: write through
<6>[ 5410.028191]  sdb: sdb1
<3>[ 5410.077497] sd 3:0:0:0: [sdb] Assuming drive cache: write through
<5>[ 5410.077500] sd 3:0:0:0: [sdb] Attached SCSI removable disk
<6>[ 5428.108189] usb 1-2: USB disconnect, address 5
```

Answer to question 3: Try to find your own forensic session processes.

The exact answer (especially the PID) will vary, but the students should be able to see their shell and the **insmod** program that loads the kernel module, perhaps the **netcat** program, if they used it to transfer the data to the analysis appliance.

A nice view can be seen from **linux_pstree** output:

```
> vol.py --plugins=../vol-profiles --profile=LinuxOpenSuSE12_2-x86x86 -f opsu.lime
linux_pstree
Volatility Foundation Volatility Framework 2.3.1
Name                 Pid             Uid
systemd              1               0
.udevd               230             0
..udevd              437             0
..udevd              438             0
.systemd-journal     239             0
.haveged             386             0
.systemd-logind      391             0
.login               402             0
..bash               1979            1000
...su                2368            1000
....bash             2369            0
.....insmod          2651            0
… [truncated] …
```

For more detail, **linux_bash** can be used:

```
> vol.py --plugins=../vol-profiles --profile=LinuxOpenSuSE12_2-x86x86 -f opsu.lime linux_bash
Volatility Foundation Volatility Framework 2.3.1
Pid      Name                 Command Time                   Command
-------- -------------------- ------------------------------ -------
```

```
… [truncated] …
    2369 bash                 2014-09-16 14:18:53 UTC+0000   make
    2369 bash                 2014-09-16 14:19:58 UTC+0000   insmod lime-3.4.6-2.10-default.ko
"path=/tmp/opsu.lime format=lime"
```

2.4.2 Task 3.2: Linux network connection analysis

The teacher will introduce the basics of Linux networking commands[62] in Volatility, namely the **linux_arp, linux_ifconfig, linux_route_cache, linux_netstat, linux_pkt_queues** and **linux_sk_buff_cache** commands. The last two commands are more complex than the others, and thus require a longer explanation.[63,64]

Commands used in this task:

```
vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux netstat -U

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_ifconfig

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_route_cache

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_arp
```

Questions to task 3.2:

1. What network ports/connections are open on the kbeast infected appliance?
2. Can you recover more information about recent network activity of this system?
3. Has one of the interfaces been put into promiscuous mode?
4. Can you find any hints, that traffic from/to the system has been redirected to other hosts?

Answer to question 1: What network ports/connections are open on the kbeast infected appliance?

Three ports: tcp/631 (on IPv4 and IPv6 localhost) and udp/68 (all addresses). Opening processes are the CUPS daemon (631) and dhclient (68). As we are looking for network activity, we use the -U flag to omit Unix socket in the **linux_netstat** plugin's output.

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_netstat -U

Volatility Foundation Volatility Framework 2.3.1

TCP      ::1:631              :::0      LISTEN             cupsd/1561

TCP      127.0.0.1:631    0.0.0.0:0     LISTEN             cupsd/1561

UDP      0.0.0.0:68       0.0.0.0:0                      dhclient/3771
```

Answer to question 2: Can you recover more information about recent network activity of this system?

[62] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#networking (last visited: 30. 9. 2014)
[63] http://vger.kernel.org/~davem/skb.html (last visited: 30. 9. 2014)
[64] http://www.linuxfoundation.org/collaborate/workgroups/networking/sk_buff (last visited: 30. 9. 2014)

As there are no open connections, we can still look for packets left in one of the receiving or sending queues with **linux_pkts**[65] and **linux_sk_buff_cache**[66]. (The command reference also explains the output format.).

```
> mkdir pkts

> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_pkt_queues -D pkts

Volatility Foundation Volatility Framework 2.3.1

Wrote 56 bytes to receive.4121.22

Wrote 56 bytes to receive.4125.22
```

Packets can be found for the processes 4121 and 4125, each on socket 22. Let's see what this is. First, the corresponding processes.

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_psaux  -p 4121,4125

Volatility Foundation Volatility Framework 2.3.1

Pid    Uid    Gid    Arguments

4121   999    999    /usr/lib/indicator-applet/indicator-applet-session --oaf-activate-
iid=OAFIID:GNOME_FastUserSwitchApplet_Factory --oaf-ior-fd=21

4125   999    999    /usr/lib/indicator-applet/indicator-applet --oaf-activate-
iid=OAFIID:GNOME_IndicatorApplet_Factory --oaf-ior-fd=36
```

Sockets can be found with **linux_lsof.**

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_lsof  -p 4121,4125

Volatility Foundation Volatility Framework 2.3.1

Pid      FD       Path

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

    4121        0 /dev/null

    4121        1 /dev/null

    4121        2 /dev/null

   … [truncated] ...

    4121       21 /home/ubuntu/.local/share/gvfs-metadata/home-b5e65f35.log

    4121       22 socket:[20506]

    4121       23 /home/ubuntu/.cache/indicator-applet-session.log

    4125        0 /dev/null

   … [truncated] …

    4125       21 /home/ubuntu/.local/share/gvfs-metadata/home-b5e65f35.log

    4125       22 socket:[20602]

    4125       23 /home/ubuntu/.cache/indicator-applet.log
```

[65] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#linux_pkt_queues (last visited: 30. 9. 2014)

[66] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#linux_sk_buff_cache (last visited: 30. 9. 2014)

We can see that the packets belong to the sockets with the inode numbers 20506 and 20602. As their sockets don't show up in the internet sockets listing, we can assume they are Unix sockets or belong to other protocols (the first is perhaps more likely).

Caveat: Some students may try to find the Unix socket with **linux_find_file**. This won't work as the plugin doesn't work with Unix sockets, and the sockets in question don't have to be named sockets, which would be the only one to show up in the file system. Further, **linux_find_file** will take quite some time with each path. Testing all pathnames of all Unix socket (from **linux_netstat**) will take much longer than the time frame of the exercise permits.

Searching the outgoing packet queues with **linux_sk_buff_**cache shows nothing:

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_sk_buff_cache -D pkts

Volatility Foundation Volatility Framework 2.3.1

INFO    : volatility.plugins.linux.slab_info: SLUB is currently unsupported.

INFO    : volatility.plugins.linux.slab_info: SLUB is currently unsupported.
```

As for the (received) packets themselves, nothing interesting catches the eye.

```
> hexdump -C receive.4121.22
00000000  6c 02 01 01 00 00 00 00  04 00 00 00 26 00 00 00  |l...........&...|
00000010  06 01 73 00 05 00 00 00  3a 31 2e 34 35 00 00 00  |..s.....:1.45...|
00000020  05 01 75 00 02 00 00 00  07 01 73 00 05 00 00 00  |..u.......s.....|
00000030  3a 31 2e 34 36 00 00 00                           |:1.46...|
00000038
> hexdump -C receive.4125.22
00000000  6c 02 01 01 00 00 00 00  07 00 00 00 26 00 00 00  |l...........&...|
00000010  06 01 73 00 05 00 00 00  3a 31 2e 35 35 00 00 00  |..s.....:1.55...|
00000020  05 01 75 00 02 00 00 00  07 01 73 00 05 00 00 00  |..u.......s.....|
00000030  3a 31 2e 34 36 00 00 00                           |:1.46...|
00000038
```

Answer to question 3: Has one of the interfaces been put into promiscuous mode?

The answer to the third question is: None, see the output from **linux_ifconfig**

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_ifconfig

Volatility Foundation Volatility Framework 2.3.1

Interface        IP Address           MAC Address        Promiscous Mode
---------------- -------------------- ------------------ ---------------

lo               127.0.0.1            00:00:00:00:00:00  False

eth0             10.0.2.15            08:00:27:02:70:c5  False
```

Answer to question 4: Can you find any hints, that traffic from/to the system has been redirected to other hosts?

The answer is (again): None. There are no signs of redirections from the ARP cache or the routing table. See the output from **linux_arp** and **linux_route_cache**.

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_route_cache

Volatility Foundation Volatility Framework 2.3.1

Interface        Destination          Gateway

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

> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_arp

Volatility Foundation Volatility Framework 2.3.1

[::                                     ] at 00:00:00:00:00:00    on lo
```

2.4.3 Task 3.3: File and kernel analysis

The teacher will introduce the following commands: **linux_lsmod** [67] , **linux_mount**[57], **linux_mount_cache**[57], **linux_dentry_cache**[57], **linux_tmpfs**[67] and **linux_bash**[61].

Commands used in this task:

```
vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux mount | grep tmpfs

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux tmpfs -L

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_tmpfs -S 4 -D cow

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_tmpfs -S 5 -D tmp
```

Questions to task 3.3:

1. Can you find something interesting in the temporary file systems?
2. What user(s)/group(s) did the interesting files/directories belong to?

Answer to question 1: Can you find something interesting in the temporary file systems?

The **linux_tmpfs** gives a listing of the temporary file systems used as well as the superblock numbers. The **linux_mount** plugin would not give the superblock numbers.

Caveat: devtmpfs is not a temporary file system.

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_mount | grep tmpfs

none             /var/run          tmpfs        rw,relatime,nosuid

none             /lib/init/rw      tmpfs        rw,relatime,nosuid

none             /dev/shm          tmpfs        rw,relatime,nosuid,nodev

tmpfs            /cow              tmpfs        rw,noatime

none             /dev              devtmpfs     rw,relatime

tmpfs            /tmp              tmpfs        rw,relatime,nosuid,nodev

none             /var/lock         tmpfs        rw,relatime,nosuid,nodev,noexec
```

[67] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#kernel-memory-and-objects (last visited: 30. 9. 2014)

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_tmpfs -L
Volatility Foundation Volatility Framework 2.3.1
1 -> /var/run
2 -> /lib/init/rw
3 -> /dev/shm
4 -> /cow
5 -> /tmp
6 -> /var/lock
```

Forensically interesting are two filesystems. First, in **tmp/tmp** (see next listing below, 2nd command) we find file **vteQ1SLDW**. This file looks like it contains a complete "listing" of the "attackers" activity on the system.

```
> head -12 vteQ1SLDW
ubuntu@ubuntu:~$ more /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=10.04
DISTRIB_CODENAME=lucid
DISTRIB_DESCRIPTION="Ubuntu 10.04.3 LTS"
ubuntu@ubuntu:~$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
aufs                    513244     89632    423612  18% /
none                    508312       244    508068   1% /dev
/dev/sr0                704226    704226         0 100% /cdrom
/dev/loop0              675968    675968         0 100% /rofs
none                    513244       248    512996   1% /dev/shm
… [truncated] …
```

Second **/cow**, which stands for Copy On Write. The image was taken from a live CD system, and as the system can't write to CD, every write is redirected into the /cow filesystem (this is called a union mount in Linux). Thus all writes not followed by a deletion of the file are preserved here.

Dumping the two file systems can be done with the **linux_tmpfs** plugin, this time with the -D option for the dump directory and -S option for the superblock number.

```
> mkdir tmp cow
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_tmpfs -S 4 -D cow
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_tmpfs -S 5 -D tmp
```

Most of the changes stem from the automatic update of several packages, the list can be seen in the directory **cow/var/cache/apt/archives/**.

In **cow/home/ubuntu/_h4x_/** we can find the source code of the rootkit as well as the compiled kernel module (ipsecs-kbeast-v1.ko) and the configuration file **config.h**.

```
> ls -l
-rw-r--r-- 1      0  1. Mai 2012  acctlog.999
-rw-r--r-- 1   3027  1. Mai 2012  bd-ipsecs-kbeast-v1.c
```

```
-rw-r--r-- 1    979  1. Mai 2012  config.h
-rwxr-xr-x 1  12282  1. Mai 2012  _h4x_bd
drwxr-xr-x 2   4096  1. Mai 2012  init
-rw-r--r-- 1  20184  1. Mai 2012  ipsecs-kbeast-v1.c
-rw-r--r-- 1  20014  1. Mai 2012  ipsecs-kbeast-v1.cc1
-rw-r--r-- 1  16420  1. Mai 2012  ipsecs-kbeast-v1.ko
-rw-r--r-- 1   1540  1. Mai 2012  ipsecs-kbeast-v1.mod.c
-rw-r--r-- 1   3592  1. Mai 2012  ipsecs-kbeast-v1.mod.o
-rw-r--r-- 1  13404  1. Mai 2012  ipsecs-kbeast-v1.o
-rw-r--r-- 1    532  1. Mai 2012  LICENSE
-rw-r--r-- 1    172  1. Mai 2012  Makefile
-rw-r--r-- 1     46  1. Mai 2012  modules.order
-rw-r--r-- 1      0  1. Mai 2012  Module.symvers
-rw-r--r-- 1   1899  1. Mai 2012  README.TXT
-rwxr-xr-x 1   5693  1. Mai 2012  setup
```

In **cow/var/log**, some traces of the “attacker” can be found in **auth.log**:

```
> grep sudo auth.log

May  1 13:15:53 ubuntu sudo:   ubuntu : TTY=unknown ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/usr/sbin/synaptic --hide-main-window --non-interactive --parent-window-id 58720295 --
update-at-startup

May  1 13:17:59 ubuntu sudo:   ubuntu : TTY=unknown ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/usr/sbin/synaptic --hide-main-window --non-interactive --parent-window-id 58720295 -o
Synaptic::closeZvt=true --progress-str Please wait, this can take some time. --finish-str
Update is complete --set-selections-file /tmp/tmpnwYMjC

May  1 14:09:00 ubuntu sudo:   ubuntu : TTY=unknown ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/usr/sbin/synaptic --hide-main-window --non-interactive --parent-window-id 58720295 -o
Synaptic::closeZvt=true --progress-str Please wait, this can take some time. --finish-str
Update is complete --set-selections-file /tmp/tmpXkSWVz

May  1 14:10:29 ubuntu sudo:   ubuntu : TTY=pts/0 ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/etc/init.d/networking restart

May  1 14:21:35 ubuntu sudo:   ubuntu : TTY=pts/0 ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/bin/kill -TERM 1350

May  1 14:22:15 ubuntu sudo:   ubuntu : TTY=pts/0 ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/bin/kill -TERM 986

May  1 14:45:13 ubuntu sudo:   ubuntu : TTY=pts/0 ; PWD=/home/ubuntu/_h4x_ ; USER=root ;
COMMAND=/sbin/insmod ipsecs-kbeast-v1.ko

May  1 14:48:34 ubuntu sudo:   ubuntu : TTY=pts/0 ; PWD=/home/ubuntu ; USER=root ;
COMMAND=/bin/netstat -tnlp

>
```

Answer to question 2: What user(s)/group(s) did the interesting files/directories belong to?

Since the plugin cannot set users on the system, the recovered files are written to, and Volatility has no access to /etc/passwd or /etc/group on the system the image was taken from, therefore user ids and group ids cannot be recovered.

2.5 Task 4: Finding and obtaining malware from memory images

In the introduction of these two tasks, the teacher will familiarize the students with plugins to find malware[68], like **malfind**, **ldrmodules**, **impscan**, **apihooks**, **driverirp**, **devicetree**, **gdt**, **idt**, **ssdt**. Emphasis should be not only on the commands and their output formats but also in the heuristics used in these commands. For **idt** and **ssdt**, some background information about the Interrupt Descriptor Table (IDT)[16,18] and the System Service Descriptor Table (SSDT)[16] may be appropriate, depending on students' level of prior knowledge. For the rest of the exercise, only **malfind** is needed.

Concerning malware recovery, the teacher should introduce the **dumpfiles** and the **filescan** plugins.

2.5.1 Task 4.1 Windows malware

Commands used:

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem malfind -D zeus.malfind --output-file=zeus-malfind

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem filescan > zeus.filescan

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem dumpfiles -D zeus.files -n

ls -l zeus.files | grep sdra

file zeus.files/file.632.0xff37f270.sdra64.exe.dat

strings windows-zeus.vmem | grep 193.104.41.75

strings -td -n 6 -a windows-zeus.vmem > zeus.s

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem strings -s zeus.s > zeus.strings

grep  193.104.41.75 zeus.stringscan
```

Answer to question 1: Where can Zeus be found in the system?

The **malfind** plugin lists all memory sections that Volatility considers hidden or injected. The output is quite long, so it is better written to a file.

```
> mkdir zeus.malfind

> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem malfind -D zeus.malfind --output-file=zeus-malfind
```

Some of these sections look like a whole executable was injected into the process. Note the MZ header.

```
Process: services.exe Pid: 676 Address: 0x7e0000

Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE

Flags: CommitCharge: 38, MemCommit: 1, PrivateMemory: 1, Protection: 6


0x007e0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............

0x007e0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......

0x007e0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x007e0030  00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00   ................


0x7e0000 4d              DEC EBP

0x7e0001 5a              POP EDX

0x7e0002 90              NOP
```

[68] https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#rootkit-detection (last visited: 30. 9. 2014)

```
0x7e0003 0003             ADD [EBX], AL
0x7e0005 0000             ADD [EAX], AL
0x7e0007 000400           ADD [EAX+EAX], AL
0x7e000a 0000             ADD [EAX], AL
… [truncated] …
```

Answer to question 2: Extract a sample of the malware from the image.

To obtain the malware, one can dump one of the sections into a file. The **-D** option of the **malfind** plugin dumps the sections to files inside the directory given with the option (**zeus.malfind** in the above example).

Another way is to look for the **sdra64.exe** executable in the memory image. The **filescan** plugin gives a listing of all open files found in the image, while the **dumpfiles** plugin allows to dump these files.

```
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem filescan > zeus.filescan
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem dumpfiles -D zeus.files -n
> ls -l zeus.files | grep sdra
-rw-r--r-- 1  1000 1000   131072 8. Sep 14:39 file.632.0xff37f270.sdra64.exe.dat
> file zeus.files/file.632.0xff37f270.sdra64.exe.dat
zeus.files/file.632.0xff37f270.sdra64.exe.dat: PE32 executable (GUI) Intel 80386, for MS
Windows
```

```
Offset(P)     #Ptr   #Hnd Access Name
---------- ------ ------ ------ ----
0x029d9b40      1      1 R----- \Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe
0x029d9cf0      1      0 -WD--- \Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe
```

Answer to question 3: Finding the C&C server download URL:

From task 2.2 we have a destination IP address: 193.104.41.75. As the destination port is 80, we can assume that it is from a HTTP connection and lets further assume the IP address is the one of the C&C server. We can use the strings plugin to search for other occurrences of the IP address in the image and maybe find a complete URL with this IP address.

A very simple way is to do this by simply using the string command, like this:

```
> strings windows-zeus.vmem | grep 193.104.41.75
://193.104.41.75/cbd/75.bro
://193.104.41.75/cbd/75.bro
193.104.41.75
: 193.104.41.75
q CKM193.104.41.75
http://193.104.41.75/cbd/75.bro
193.104.41.75
```

With Volatility's **strings**[69] plugin, we can extract a little more information:

[69] https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#strings (last visited: 30. 9. 2014)

```
> strings -td -n 6 -a windows-zeus.vmem > zeus.s
> vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem strings -s zeus.s > zeus.strings
> grep  193.104.41.75 zeus.stringscan
0072eca4 [632:75b2eca4 688:7612eca4 856:000c3ca4 1668:2012eca4] ://193.104.41.75/cbd/75.bro
0072eebc [632:75b2eebc 688:7612eebc 856:000c3ebc 1668:2012eebc] ://193.104.41.75/cbd/75.bro
0400ece8 [856:000e1ce8] 193.104.41.75
05b2b3fc [856:000c63fc] : 193.104.41.75
05b2b7e7 [856:000c67e7] q CKM193.104.41.75
07617d98 [856:00c32d98] http://193.104.41.75/cbd/75.bro
07617dd8 [856:00c32dd8] 193.104.41.75
```

The extra effort gives more precise information. We do have not only the strings, but also the processes (PIDs) and virtual addresses (in the processes address space) of the strings (see command reference to **strings**).

2.5.2 Task 4.2 Linux malware

In the introductory part of this task, the teacher will familiarize the students with the basic commands used to find malware in a Linux image.

linux_check_idt and **linux_check_syscall** are very similar in concept to the **cidt** and **ssdt** commands for Windows, as is **linux_check_modules** to **modules** or **modscan**.

There are several places in a Linux system, where malware can manipulate function pointers inside the kernel, the **linux_check_afinfo, linux_check_tty, linux_keyboard_notifier**, and **linux_check_fop** commands are used to check these.

Special emphasis should be given to the explanation of the **linux_check_creds** commands as it works differently from the other commands in this task. However, it will not be used in the exercise.

Finally, the teacher will introduce the **linux_moddump** command.

Commands used

```
vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_lsmod

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-clean.mem linux_check_syscall > syscall.clean

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux check syscall > syscall.kbeast

diff -y --suppress-common-lines syscall.clean syscall.kbeast

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_check_afinfo
```

Answer to question 1: Where/how can the kbeast rootkit be found?

KBeast is implemented as a Linux kernel module that hides itself by unlinking itself from the list of modules exported through the /proc/modules interface. Consequently, it does not show on **linux_lsmod**. It will be found with **linux_check_modules**, which uses the **/sys/module/** interface.

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_lsmod

Volatility Foundation Volatility Framework 2.3.1

>
```

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_check_modules

Volatility Foundation Volatility Framework 2.3.1

Module Name

-----------

ipsecs_kbeast_v1

… [truncated] …
```

Answer to question 2: What functions have been manipulated by the rootkit?

To find a list of syscalls manipulated by malware, the **linux_check_syscalls** plugin can be used. Syscalls that Volatility believes to have been manipulated are marked as "**HOOKED**".

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_check_syscall > syscall.kbeast

Table Name      Index Address    Symbol

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

32bit             0x0 0xc015d0c0 sys_restart_syscall

32bit             0x1 0xc0151520 sys_exit

32bit             0x2 0xc0103560 ptregs_fork

32bit             0x3 0xf817d870 HOOKED

32bit             0x4 0xf817e070 HOOKED

32bit             0x5 0xf817e600 HOOKED

32bit             0x6 0xc0207dd0 sys_close

… [truncated] …
```

Since the output is quite long, it is advisable to redirect to a file (see listing above). If a clean baseline is present (i.e. from **Ubuntu-10.04.3-i386-LiveCD-clean.mem**), both **syscall** lists can be compared to find exactly which calls have been manipulated. One can use the Gnu **diff** command for this (as the list is not too long, it can be done by hand in reasonable time).

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-kbeast.mem linux_check_syscall > syscall.kbeast

> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-clean.mem linux_check_syscall > syscall.clean

> diff -y --suppress-common-lines syscall.clean syscall.kbeast

32bit     0x3 0xc020a820 sys_read          | 32bit       0x3 0xf817d870 HOOKED

32bit     0x4 0xc020a7b0 sys_write         | 32bit       0x4 0xf817e070 HOOKED

32bit     0x5 0xc0208030 sys_open          | 32bit       0x5 0xf817e600 HOOKED

32bit     0xa 0xc0215e80 sys_unlink        | 32bit       0xa 0xf817e1c0 HOOKED

32bit    0x25 0xc015f670 sys_kill          | 32bit      0x25 0xf817d520 HOOKED

32bit    0x26 0xc0215cd0 sys_rename        | 32bit      0x26 0xf817e470 HOOKED

32bit    0x28 0xc0215fd0 sys_rmdir         | 32bit      0x28 0xf817e2a0 HOOKED

32bit    0x81 0xc0180d20 sys_delete_module | 32bit      0x81 0xf817e6f0 HOOKED

32bit    0xdc 0xc0218ff0 sys_getdents64    | 32bit      0xdc 0xf817e7c0 HOOKED

32bit    0x12d 0xc0215f90 sys_unlinkat     | 32bit     0x12d 0xf817e380 HOOKED
```

Likewise, the IDT can be checked with **linux_check_idt**. However, the entries marked as hooked are also marked in the clean image, so they are likely to be false positives.

Likewise, the tty devices are not manipulated, **linux_check_tty** reports the same addresses for the clean and the infected image[70].

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-clean.mem linux_check_tty

Volatility Foundation Volatility Framework 2.3.1

Name              Address    Symbol

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

tty1              0xc03bade0 n_tty_receive_buf

tty2              0xc03bade0 n_tty_receive_buf

tty3              0xc03bade0 n_tty_receive_buf

tty4              0xc03bade0 n_tty_receive_buf

tty5              0xc03bade0 n_tty_receive_buf

tty6              0xc03bade0 n_tty_receive_buf

tty8              0xc03bade0 n_tty_receive_buf

> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_check_tty

Volatility Foundation Volatility Framework 2.3.1

Name              Address    Symbol

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

tty1              0xc03bade0 n_tty_receive_buf

tty2              0xc03bade0 n_tty_receive_buf

tty3              0xc03bade0 n_tty_receive_buf

tty4              0xc03bade0 n_tty_receive_buf

tty5              0xc03bade0 n_tty_receive_buf

tty6              0xc03bade0 n_tty_receive_buf

tty8              0xc03bade0 n_tty_receive_buf
```

As there are no processes that are manipulated by the rootkit, **linux_check_creds** does not report anything.

Lastly, there are the file operation pointers[71] (**linux_check_fop**) and the network protocol function pointers (**linux_check_afinfo**). The later gives another manipulated function:

```
> vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-
LiveCD-kbeast.mem linux_check_afinfo

Volatility Foundation Volatility Framework 2.3.1

Symbol Name                          Member                         Address

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

tcp4_seq_afinfo                      show                           0xf817d650
```

Answer to question 3: Can you extract a sample of the malware from the image?

The linux_check_modules plugin reports a hidden kernel module (see listing above). However, as of Volatility 2.3.1, only modules present in the module list (i.e. which show up on linux_lsmod) can be dumped with **linux_moddump**. So, the answer to this question is: no.

[70] http://www.ieee.security.org/TC/SPW2013/proceedings/4740a097.pdf (last visited: 30. 9. 2014)
[71] http://volatility-labs.blogspot.com/2012/09/movp-14-average-coder-rootkit-bash.html (last visited: 30. 9. 2014)

3 Commands used

```
TASK 1.1 (Acquiring a Windows memory image)
winpmem 1.4.exe testdump.raw
dir testdump.raw

netcat -l -p 11111 > testdump.crash          # on the receiving system
winpmem 1.4.exe -d - | ncat.exe --send-only exercise.example.net 11111
ls -l testdump.crash
vol.py -f testdump-net.raw imageinfo

TASK 1.2.1 (Building the LiME memory extraction module)
cd src
make
uname -r
ls -l

TASK 1.2.2 (Acquiring a Linux memory image)
# remember that insmod requires root privileges!
insmod lime-3.4.6-2.10-default.ko "path=/tmp/testdump.lime format=lime"
ls -l /tmp/testdump.lime

TASK 1.2.3 (Building a Linux profile)
cd volatility-tools/linux
make
zip OpenSuSE12.2-x86.zip module.dwarf /boot/System.map-3.4.6-2.10-default

TASK 2.1 (Windows process analysis)
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem pslist
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem dlllist -p 856
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem envars -p 856
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem privs -p 856
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem getsids -p 856

TASK 2.2 (Windows network analysis)
vol.py --profile=WinXPSP2x86 -f zeus.vmem connections
vol.py --profile=WinXPSP2x86 -f zeus.vmem connscan
vol.py --profile=WinXPSP2x86 -f zeus.vmem sockets
vol.py --profile=WinXPSP2x86 -f zeus.vmem sockscan

TASK 2.3 (Windows Registry analysis)
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem hivelist
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem userassist
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -K "Microsoft\Windows
NT\CurrentVersion\Winlogon"
```

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -K "Microsoft\Windows
NT\CurrentVersion\Winlogon"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem printkey -o 0xe101b008 -K
"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile"

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem hashdump -y 0xe101b008 -s 0xe1544008


TASK 2.4 (Windows file and other analysis)

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem handles -t File | grep sdra

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem pstree

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem handles -p632,676,856 -t File,Mutant | grep
-i avira

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem consoles

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem cmdscan


TASK 3.1 (Linux process analysis)

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_cpuinfo

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_dmesg

vol.py --plugins=../vol-profiles --profile=LinuxOpenSuSE12 2-x86x86 -f opsu.lime linux pstree

vol.py --plugins=../vol-profiles --profile=LinuxOpenSuSE12 2-x86x86 -f opsu.lime linux bash


TASK 3.2 (Linux network analysis)

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_netstat -U

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux ifconfig

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_route_cache

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux arp


TASK 3.3 (Linux file analysis)

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux mount | grep tmpfs

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux tmpfs -L

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_tmpfs -S 4 -D cow

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux tmpfs -S 5 -D tmp


TASK 4.1 (Finding and obtaining Windows malware)

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem malfind -D zeus.malfind --output-file=zeus-
malfind
```

```
vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem filescan > zeus.filescan

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem dumpfiles -D zeus.files -n

ls -l zeus.files | grep sdra

file zeus.files/file.632.0xff37f270.sdra64.exe.dat

strings windows-zeus.vmem | grep 193.104.41.75

strings -td -n 6 -a windows-zeus.vmem > zeus.s

vol.py --profile=WinXPSP2x86 -f windows-zeus.vmem strings -s zeus.s > zeus.strings

grep  193.104.41.75 zeus.stringscan


TASK 4.2 (Finding and obtaining Linux malware)

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_lsmod

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
clean.mem linux check syscall > syscall.clean

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10 04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux check syscall > syscall.kbeast

diff -y --suppress-common-lines syscall.clean syscall.kbeast

vol.py --plugins=../vol-profiles --profile=LinuxUbuntu10_04x86 -f ubuntu-10.04.3-i386-LiveCD-
kbeast.mem linux_check_afinfo
```

4 Bibliography

1. Volatility Foundation http://www.volatilityfoundation.org/ (accessed: 24. October 2014)
2. ENISA Exercise no. 24 https://www.enisa.europa.eu/activities/cert/support/exercise, exercise no 24 (accessed: 24. October 2014)
3. ENISA Background information on CERTs and CSIRTs https://www.enisa.europa.eu/activities/cert/background (accessed: 24. October 2014)
4. ENISA Support for CERTs / CSIRTs https://www.enisa.europa.eu/activities/cert/support (accessed: 24. October 2014)
5. NIST Computer Security Incidents Handling Guide http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf (accessed: 24. October 2014)
6. NIST Guide to Integrating Forensic Techniques into Incident Response http://csrc.nist.gov/publications/nistpubs/800-86/SP800-86.pdf (accessed: 24. October 2014)
7. Mitchell: *Concepts in Programming Languages*, Cambridge University Press, ISBN 0521780985
8. Van Roy, et al.: *Concepts, Techniques, and Models of Computer Programming*, MIT Press, ISBN: 9780262220699
9. C Programming and C++ Programming http://www.cprogramming.com/(accessed: 24. October 2014)
10. Wikibooks: C Programming http://en.wikibooks.org/wiki/C_Programming (accessed: 24. October 2014)
11. Wikibooks: x86 Disassembly http://en.wikibooks.org/wiki/X86_Disassembly (accessed: 24. October 2014)
12. Silberschatz, et al.: *Operating System Concepts*, 9th ed., Wiley, ISBN-13: 978-1118063330
13. Russinovich, et al.: *Windows Internals,* 6th ed., Microsoft Press, ISBN-13: 978-0735648739 (part1), ISBN-13: 978-0735665873 (part 2), http://technet.microsoft.com/de-de/sysinternals/bb963901.aspx (accessed: 24. October 2014)
14. Wolfgang Mauerer: *Professional Linux Kernel Architecture*, Wrox, ISBN-13: 978-0470343432
15. Bovet, et al.: *Understanding the Linux Kernel*, 3rd ed., O'Reilly, ISBN-13: 978-0596005658

16. Python https://www.python.org/ (accessed 24. October 2014)
17. PyCrypto – The Python Cryptography Toolkit https://www.dlitz.net/software/pycrypto (accessed 24. October 2014)
18. Python Imaging Library (PIL) http://www.pythonware.com/products/pil (accessed 24. October 2014)
19. diStrom https://code.google.com/p/distorm (accessed 24. October 2014)
20. Yara in a nutshell https://plusvic.github.io/yara/ (accessed 24. October 2014)
21. Volatility https://sourceforge.net/projects/volatility.mirror/files/winpmem-1.4.1.zip/ (accessed 24. October 2014)
22. GitHub 504ensicsLabs/Lime https://github.com/504ensicsLabs/LiME (accessed 24. October 2014)
23. GNU Binutils https://www.gnu.org/software/binutils/ (accessed 24. October 2014)
24. MinGW – Minimalist GNU for Windows http://www.mingw.org/ (accessed 24. October 2014)
25. Cygwin https://www.cygwin.com/ (accessed 24. October 2014)
26. Windows Sysinternals: Strings v2.52 http://technet.microsoft.com/en-us/sysinternals/bb897439 (accessed 24. October 2014)
27. GNU Grep https://www.gnu.org/software/grep/ (accessed 24. October 2014)
28. http://nmap.org/dist/ncat-portable-5.59BETA1.zip (accessed 27. October 2014)
29. Mandiant: Find Evil in Live Memory https://www.mandiant.com/resources/download/memoryze(accessed 24. October 2014)
30. Second Look®: Advanced Linux Malware Detection & Investigation http://secondlookforensics.com/ (accessed 24. October 2014)
31. Second Look: Linux Memory Forensics for Incident Response and Intrusion Detection https://secondlookforensics.com/linux-memory-images/ (accessed 24. October 2014)
32. http://malwarecookbook.googlecode.com/svn-history/r26/trunk/17/1/zeus.vmem.zip (accessed 27. October 2014)
33. GitHub: Volatility Usage https://github.com/volatilityfoundation/volatility/wiki/Volatility-Usage (accessed: 24. October 2014)
34. Wikipedia: Slab allocation http://en.wikipedia.org/wiki/Slab_allocation (accessed: 24. October 2014)
35. Notes on computer forensics – international edition: Scanning for File Objects http://computer.forensikblog.de/en/2009/04/scanning-for-file-objects.html (accessed: 24. October 2014)
36. GitHub: Address Spaces https://github.com/volatilityfoundation/volatility/wiki/Address-Spaces (accessed: 24. October 2014)
37. GitHub: Firewire Address Space https://github.com/volatilityfoundation/volatility/wiki/Firewire-Address-Space (accessed: 24. October 2014)
38. GitHub: VMware Snapshot File https://github.com/volatilityfoundation/volatility/wiki/VMware-Snapshot-File (accessed: 24. October 2014)
39. GitHub: Virtual Box Core Dump https://github.com/volatilityfoundation/volatility/wiki/Virtual-Box-Core-Dump (accessed: 24. October 2014)
40. GitHub: Crash Address Space https://github.com/volatilityfoundation/volatility/wiki/Crash-Address-Space (accessed: 24. October 2014)
41. GitHub: Hiber Address Space https://github.com/volatilityfoundation/volatility/wiki/Hiber-Address-Space (accessed: 24. October 2014)
42. GitHub: Command Reference: Image Info https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#imageinfo (accessed: 24. October 2014)

43. CONFIG_STRICT_DEVMEM: Filter access to /dev/mem http://cateee.net/lkddb/web-lkddb/STRICT_DEVMEM.html (accessed: 24. October 2014)
44. GitHub: Command Reference: Processes and DLLs https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#processes-and-dlls (accessed: 24. October 2014)
45. GitHub: Command Reference: Networking https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#networking (accessed: 24. October 2014)
46. GitHub: Command Reference: Registry https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#registry (accessed: 24. October 2014)
47. GitHub: Command Reference: File System https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#file-system (accessed: 24. October 2014)
48. GitHub: Command Reference: System Information https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#system-information(accessed: 24. October 2014)
49. Wikipedia: Slab https://en.wikipedia.org/wiki/SLAB (accessed: 24. October 2014)
50. Wikipedia: Slub https://en.wikipedia.org/wiki/SLUB (accessed: 24. October 2014)
51. Wikipedia: SlOB https://en.wikipedia.org/wiki/SLOB (accessed: 24. October 2014)
52. GitHub: Command Reference: Processes https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#processes (accessed: 24. October 2014)
53. The socket buffer http://vger.kernel.org/~davem/skb.html (accessed: 24. October 2014)
54. Linux Foundation: sk_buff http://www.linuxfoundation.org/collaborate/workgroups/networking/sk_buff (accessed: 24. October 2014)
55. GitHub: Command Reference: Linux pkt queues https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#linux_pkt_queues(accessed: 24. October 2014)
56. GitHub: Command Reference: Linux sk_buff cache https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#linux_sk_buff_cache(accessed: 24. October 2014)
57. GitHub: Command Reference: Kernel Memory and Objects https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#kernel-memory-and-objects (accessed: 24. October 2014)
58. GitHub: Command Reference: Rootkit Detection https://github.com/volatilityfoundation/volatility/wiki/LinuxCommand%20Reference#rootkit-detection (accessed: 24. October 2014)
59. GitHub: Command Reference: Strings https://github.com/volatilityfoundation/volatility/wiki/Command%20Reference#strings (accessed: 24. October 2014)
60. http://www.ieee.security.org/TC/SPW2013/proceedings/4740a097.pdf (accessed: 30. September 2014)
61. Volatility Labs: MoVP 1.4 Average Coder Rootkit, Bash History, and Elevated Processes http://volatility-labs.blogspot.com/2012/09/movp-14-average-coder-rootkit-bash.html (accessed: 24. October 2014)

ENISA
European Union Agency for Network and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office
1 Vass. Sofias & Meg. Alexandrou
Marousi 151 24, Athens, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu