Windows Memory Forensics with Volatility

Andreas Schuster

About the Tutorial

About the Tutorial
Agenda

Part 1– Refresher

- Memory fundamentals
- Memory acquisition techniques
- Kernel objects
- Memory analysis techniques

Part 2 – Using Volatility

- Volatility overview
- Built-in functions
- Selected plug-ins
- Hands-on exercises

Part 3 – Programming

- Address spaces
- Objects and Profiles
- Your first plug-in
- Building blocks

About the Tutorial

Acknowledgements

- **Pär Österberg Medina**
  Swedish IT Incident Center SITIC,
  Stockholm, Sweden
  http://www.sitic.se/
- **AAron Walters**
  Volatile Systems LLC, USA
  https://www.volatilesystems.com/
- **Brendan Dolan-Gavitt**
  Georgia Institute of Technology,
  Atlanta, GA, USA
  http://moyix.blogspot.com/
- **Bradley Schatz**
  Schatz Forensic Pty Ltd,
  Brisbane, Australia
  http://www.schatzforensic.com.au/
- **hogfly**
  http://forensicir.blogspot.com/

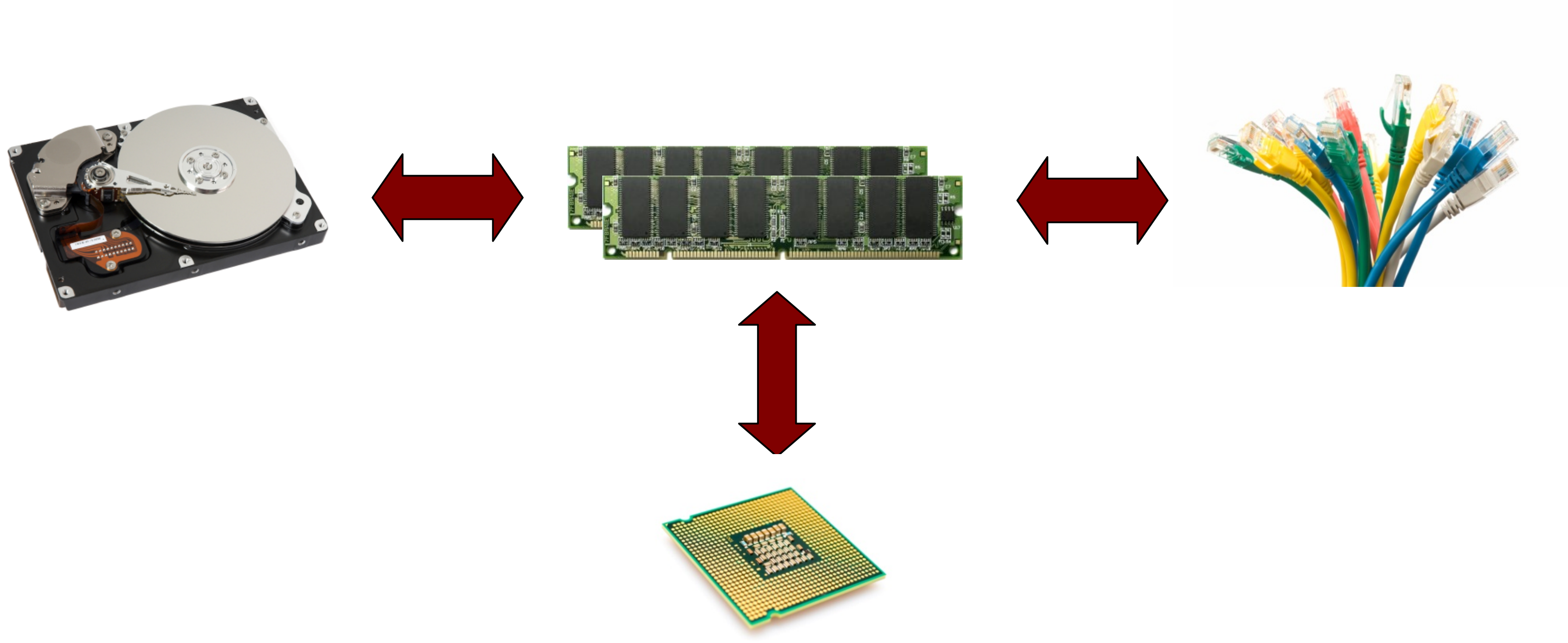
About the Tutorial
Course Materials

- Virtual machine, requires VMware player/workstation 6.5.2
  - Ubuntu Linux
  - Login as **user**, password is **us3rpw**
  - Volatility and plug-ins installed
  - Several other memory analysis tools (PTFinder, PoolTools)
  - Sample memory images
- Tools
  - VMWare Player 2.5.2 for Windows and Linux (.rpm)
  - Symbol viewers
  - Volatility 1.3.1 beta and SVN, with plug-ins
- Literature
- Slides (will be uploaded to the conference website after the tutorial)

Part 1
Memory Analysis Primer

Introduction

Why do we need Memory Analysis?



Main memory contains evidence!

Introduction
Why do we need Memory Analysis?

- No one would exclude a disk from a forensic examination. Physical memory is a storage media like a hard disk drive. So why act arbitrarily?
- Physical memory contains unique data, not just a duplicate of data that can be found elsewhere.
- When examining a network-based attack, physical memory provides the missing link between network data (capture/IDS alert) and possible artifacts on a disk.
- Only (physical) memory documents the current status of a computer/device.
- Some attacks don’t leave traces on disk, but only in memory.

Introduction
Live Response vs. Memory Analysis

Live Response

- Focus on “time”
- Acquisition and analysis in one step
    - Untrusted environment
    - Not repeatable
- Tools tend to be obtrusive

Order of Volatility
Live Response vs. Memory Analysis

Action	% RAM unchanged	
	256 MB RAM	512 MB RAM
Start	100.0	100.0
Idle for 1 hour	90.4	96.7
Idle for 2 hours	79.7	96.1
DD (live acquisition)	76.9	89.8
Idle for 15 hours	74.8	85.6
WFT (live response)	67.2	69.4

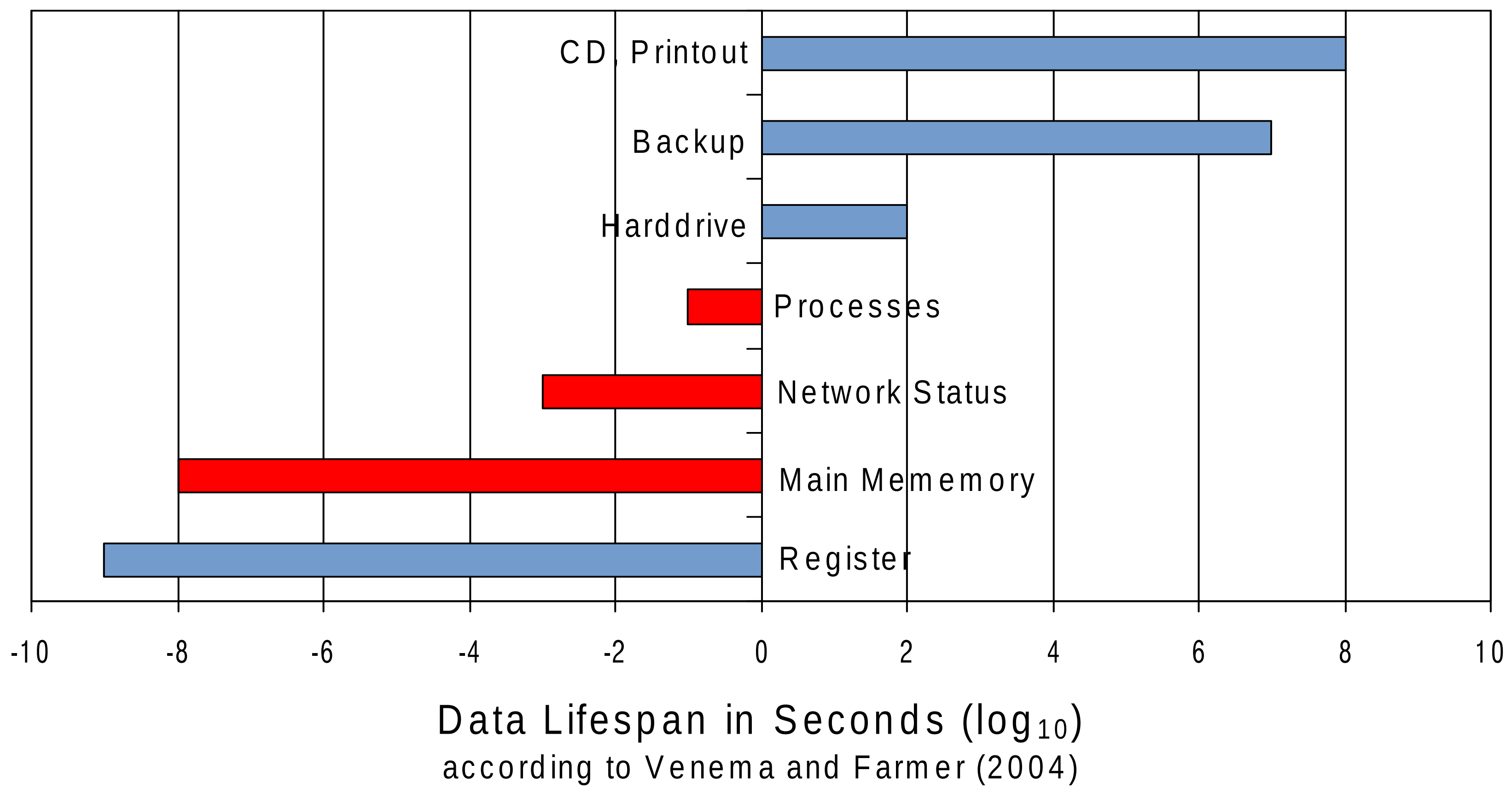
Effects on main memory, according to Walters and Petroni (2006)

Introduction
Live Response vs. Memory Analysis

Memory Analysis

- Focus on “best evidence”
- Acquisition and analysis in separate steps
    - Acquisition in an untrusted environment
    - Analysis in a trusted environment
    - Analysis tools not limited by target OS
    - Analysis is repeatable (acquisition is not)

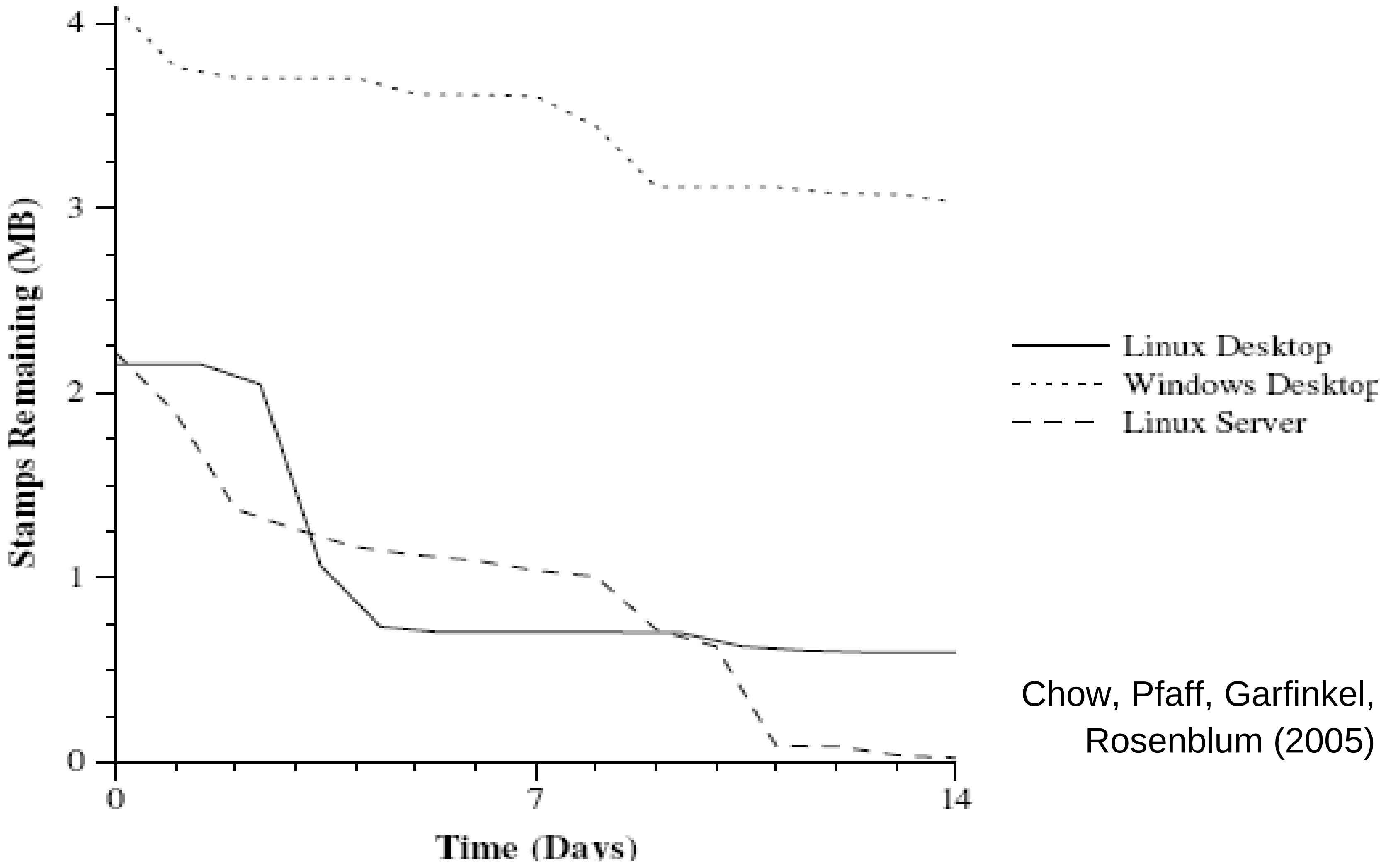
Introduction
Preserve Data in Order of Volatility



Introduction
Persistence in Userland

- Solomon, Huebner, Bem and Szeżynska (2007)
  - Age of deallocated pages does NOT affect the order of reallocation
  - Majority of pages persisted for less than 5 minutes

Introduction
Persistence in Kernel Space



Chow, Pfaff, Garfinkel,
Rosenblum (2005)

Introduction
Persistence in Kernel Space

- Schuster (2008)
  - 90% of freed process objects after 24 hours of idle activity
  - Re-allocation of memory by size, LIFO principle
  - Kernel tries to free memory pages
  - Important objects (processes, threads, files, …) are of fixed size.

- Live response can be devastating!
- Install agents prior to the incident!

Memory Acquisition

Memory Acquisition
Considerations

- Time of installation
  prior to incident vs. post incident
- Access to system
  local vs. remote
- Access to main memory
  pure hardware vs. software
- Required privileges
  user vs. administrator
- Impact on system
  in vivo vs. post mortem
- Atomicity of image
- Image file format
  - raw
  - crash dump
  - hiberfil.sys
  - EWF, AFF

Image File Formats
Raw

- “dd format”
- 1:1 copy of physical memory. Some regions may not be accessible, tough.
- offset == physical address
- Several proof-of-concept tools only operate on this format.

Image File Formats
Crash Dump

- Required by Microsoft Tools
- Extension .DMP
- CPU state information
- Segmented format:
  - One or many blocks of physical memory
  - Holes, e.g. BIOS, DMA, AGP video
  - Extra data from devices that employ `nt!KeRegisterBugCheckReasonCallback`

Image File Formats
Hibernate File

Hibernate file

- hiberfil.sys
- Compressed
- Contains only physical memory that is “in use”

Image File Formats
Expert Witness Format

- Popular, thanks to Guidance Software's EnCase and WinEn (.E01)
- libewf
  by Joachim Metz
  http://sourceforge.net/projects/libewf/
- Different levels of compression
- Meta-Information (case number, examiner, MD5 hash, etc.)
- Similar, but open source: Advanced Forensic Format (AFF)
  http://www.afflib.org/

Tools
Validation

- There's a plenty of memory acquisition tools available…
- … but none has been validated yet.
- FAIL:
  - Image of expected size, but first 256 MBytes all zero
  - Image of expected size, but repeatedly filled with first 256 MBytes
  - Page 0 missing from image

Tools
Recommendations

- VMware
  - Suspend VM, then copy “physical memory” file (.vmem)
  - Malware can (and does!) detect the hypervisor
- win32dd
  - by Mathieu Suiche
    http://win32dd.msuiche.net/
  - Free, open source
  - Produces images in either raw or crash dump formats
- kntdd
  - by George Garner Jr.
    http://www.gmgsystemsinc.com/knttools/
  - Commercial
  - Produces raw and crash dump at the same time
  - Enterprise version available (agent, X.509 certificates, etc.)

Tools
Recommendations

- F-Response
  - http://www.f-response.com/
  - Enables access to physical memory over iSCSI
  - Use with acquisition tool of your choise
- Hibernation
  - Built-in, commonly activated on laptop computers
  - `powercfg /hibernate on`
  - Cause system to hibernate, then acquire hard disk and extract hiberfil.sys
- Crash Dump
  - Built-in
  - Needs to be configured in advance, reboot required
  - Kernel dumps are small
  - Minidumps are essentially useless for forensic memory analysis

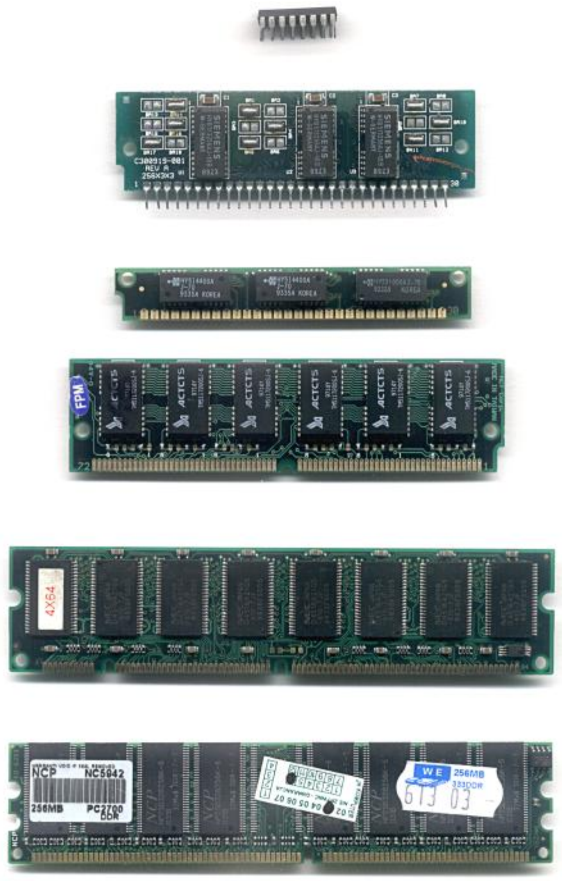
Tools
Recommendations

- FireWire
  - Read (and write!) access to lower 4 GB of physical memory
  - Python tools available at http://storm.net.nz/projects/16
  - Rutkowska (2007) redirects access to physical memory!
- Cold Boot Attack
  - Exploits remanence of DRAM
  - Cooling slows down the degradation of memory contents
  - http://citp.princeton.edu/memory/

Concepts

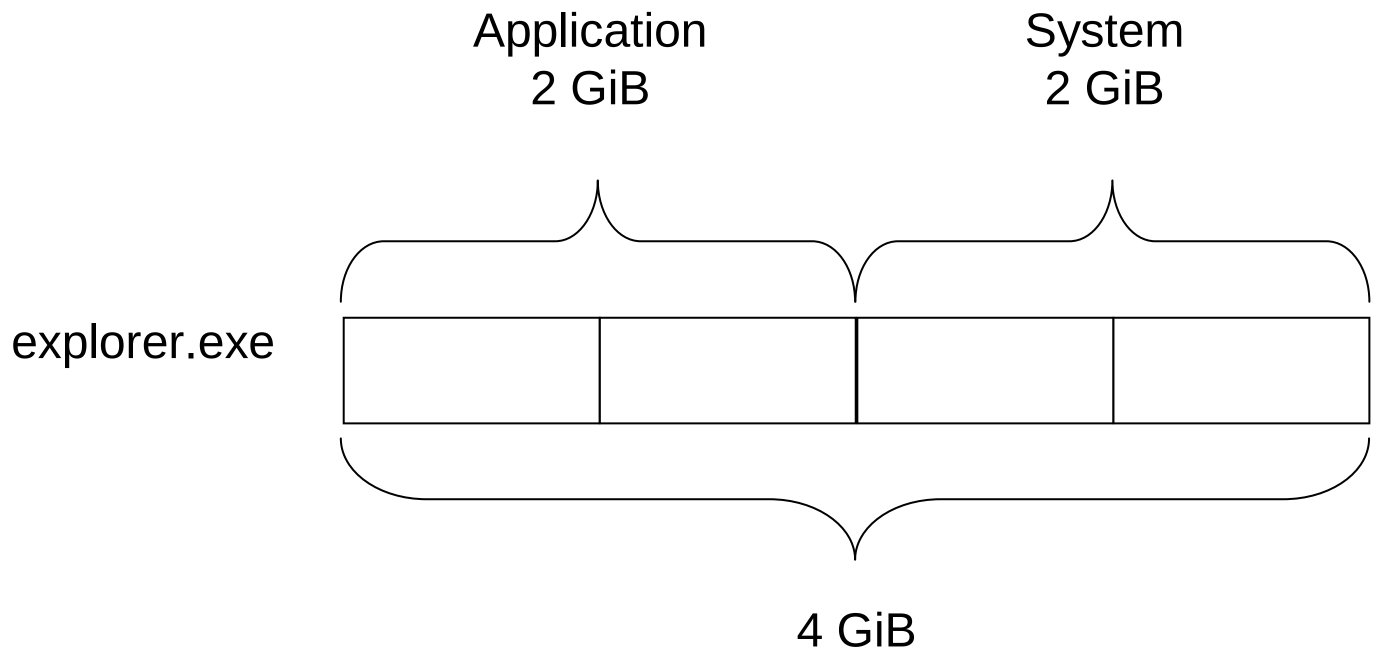
Concepts
Physical Memory

- Physical memory is the short-term memory of a computer.
- Rapid decay of information as soon as memory module is disconnected from power and clock sources.

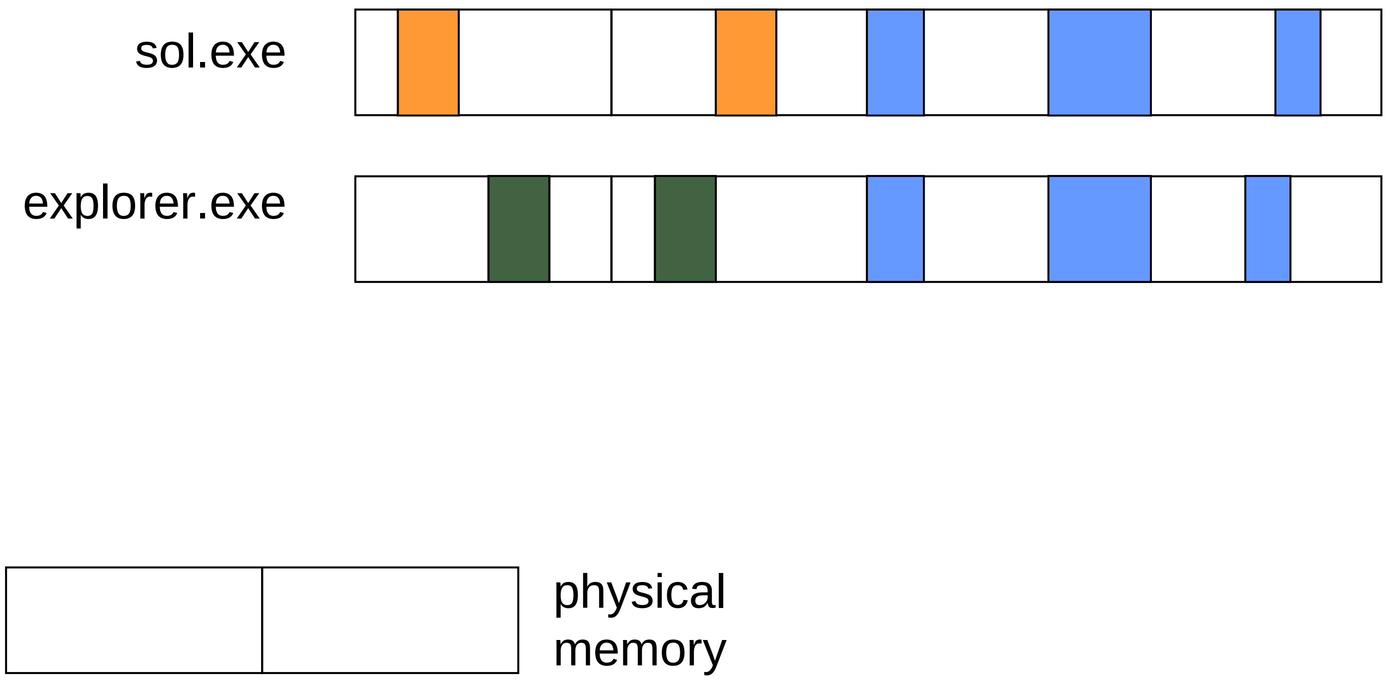


Concepts
Address Space

- 4 GiB of (virtual) address space per process
- Split into halves

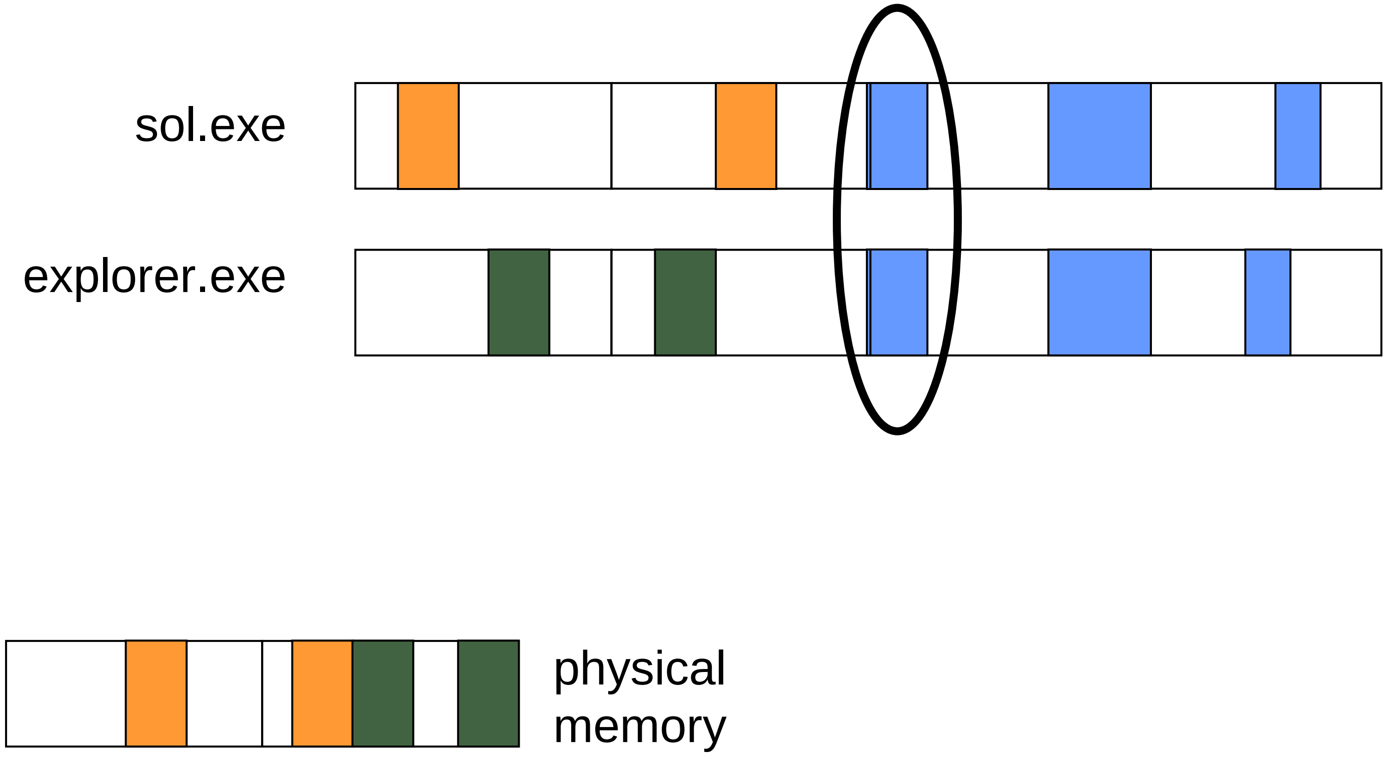


- Physical memory is divided into so called “pages”.
- Allocated virtual memory is mapped onto physical memory page by page.



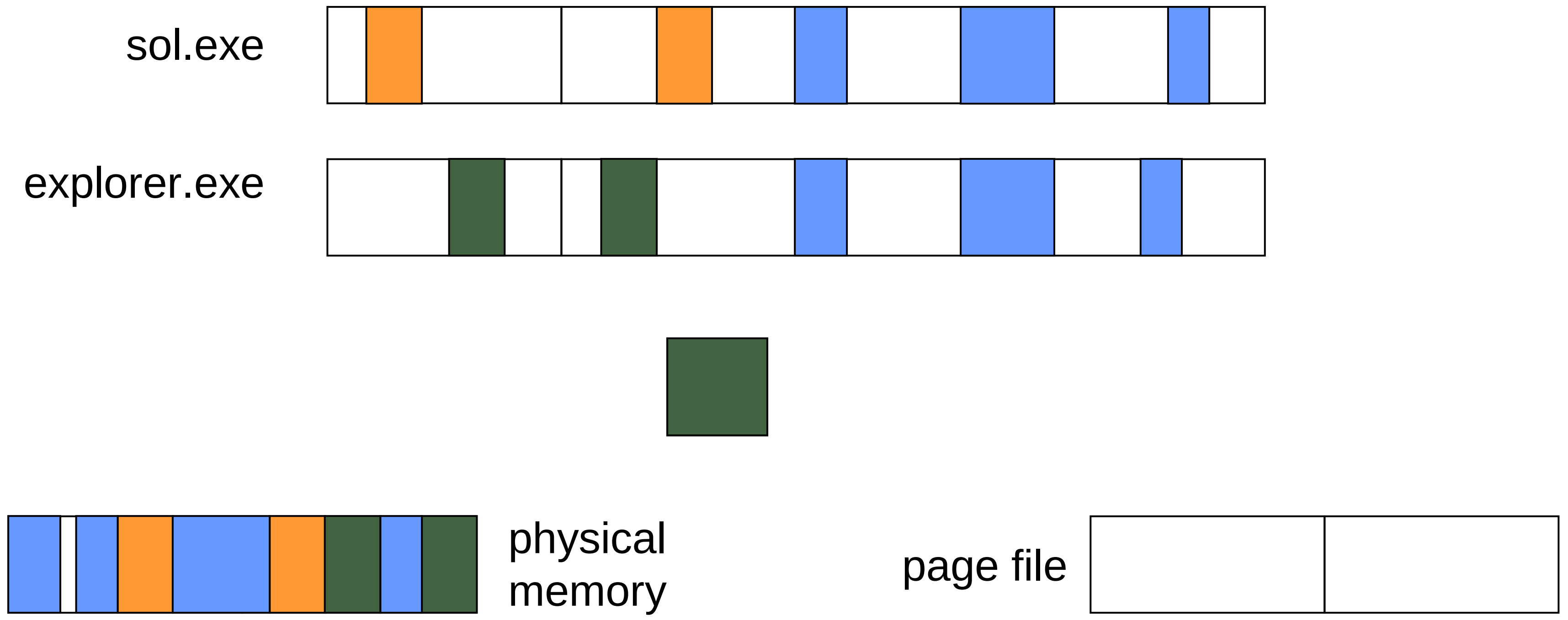
Concepts
Virtual Memory

The same page of physical memory can appear at different locations within the same address space or in different address spaces.



Concepts
Virtual Memory

Data can be moved from physical memory into a page file to clear some space.



Memory Pools

Memory Pools
Concept

- Memory is managed through the CPU’s Memory Management Unit (MMU).
- Allocation granularity at the hardware level is a whole page (usually 4 kiB).
- Concept of “pools”: several pages are pre-allocated to form a pool of memory.
- Small requests are served from the pool, granularity 8 Bytes (Windows 2000: 32 Bytes).
- There are mostly two pools:
  - non-paged pool (frequently used information like processes, threads)
  - paged-pool (allocations also can be found in page file)

Memory Pools
POOL_HEADER

```
struct _POOL_HEADER, 9 elements, 0x8 bytes
   +0x000 PreviousSize     : Bitfield Pos 0, 9 Bits
   +0x000 PoolIndex        : Bitfield Pos 9, 7 Bits
   +0x002 BlockSize        : Bitfield Pos 0, 9 Bits
   +0x002 PoolType         : Bitfield Pos 9, 7 Bits
   +0x000 Ulong1           : Uint4B
   +0x004 ProcessBilled    : Ptr32 to struct _EPROCESS
   +0x004 PoolTag          : Uint4B
   +0x004 AllocatorBackTraceIndex : Uint2B
   +0x006 PoolTagHash      : Uint2B
```

Note: There are multiple interpretations for the DWORD at offset 4.

Memory Pools
POOL_HEADER

- BlockSize:
  - size of this allocation
  - pointer to next allocation

- PreviousSize:
  - size of the previous allocation
  - pointer to previous allocation
  - must be 0 for the first allocation in a memory page

- Both:
  - measured in units of 8 bytes (Windows 2000: 32 bytes).
  - includes the _POOL_HEADER (8 bytes), so must be 1 at least.

Memory Pools
POOL_HEADER

- Pool type:
  - Declared in Windows Development Kit, file wdm.h
  - values found in memory are increased by 1
  - 0 now indicated a “free” block
  - odd value = non-paged pool
  - even value = paged pool

.

Memory Pools
POOL_HEADER

- PoolTag:
  - According to documentation of `ExAllocatePoolWithTag` in MSDN:
    - → up to 4 character literals
    - → ASCII values between 0 and 127
    - → stored in little-endian (reverse) byte-order
      ‘1234’ stored as ‘4321’
    - → every allocation code path should use a unique pool tag
    - → “protection” bit for kernel objects
  - There is no registry for pool tags.
  - Every application is free to use any pool tag!

Kernel Objects

Objects
Concept

- NT and Vista kernels are object oriented
- Uniform way to access different kinds of system resources
- Charge processes for their object (= resource) usage
- Objects can be found at different levels
    - These objects do not interoperate!
    - e.g. GDI Object (brush) and Executive Object (process)

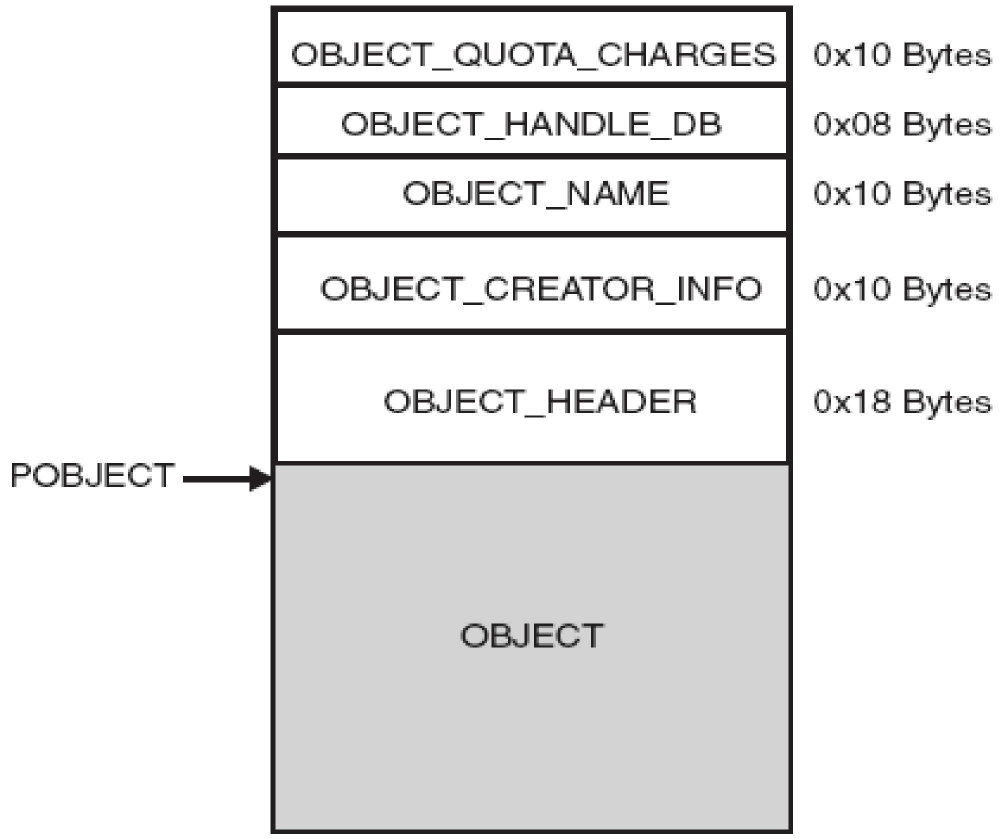
Objects
Objects of the Executive

- The Executive implements
  - 27 object types on Windows 2000
  - 29 object types on XP and Server 2003

- Important object classes
  - Thread: executable entity within a process
  - Process: execution environment, collection of ressources
  - Driver: loadable kernel module
  - File: instance of an open file or I/O device
  - Token: SID and privileges
  - Key: registry

Objects
Objects of the Executive

- All objects of the Executive share a common structure, the _OBJECT_HEADER
- Caveats
  - A pointer will always point right behind the header
  - The header grows in the direction of lower addresses



Source: Schreiber, 2001

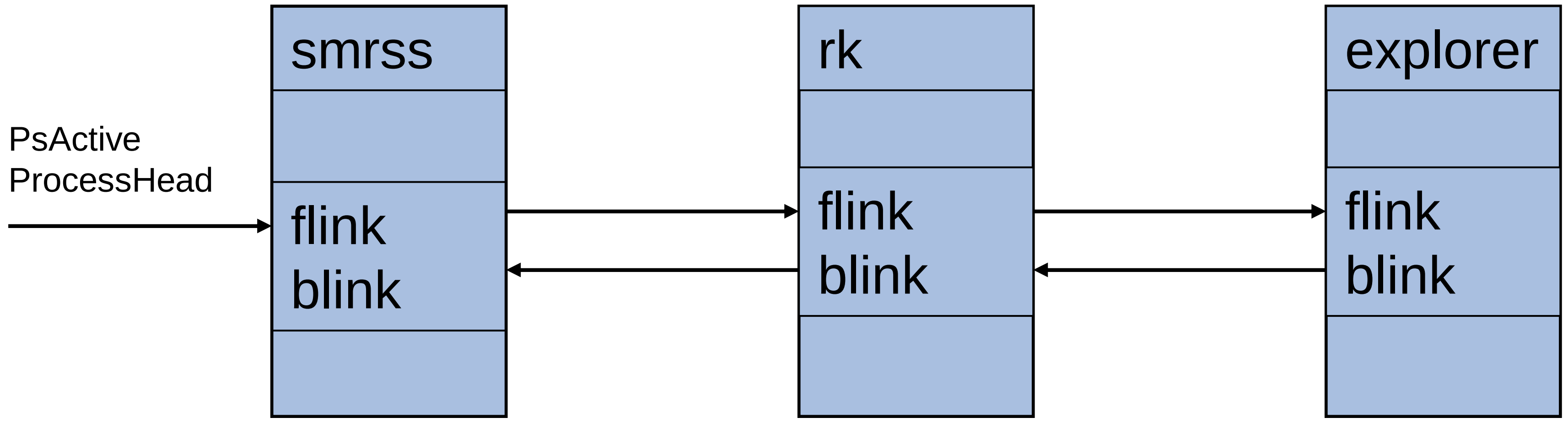
Analysis Techniques

Analysis Techniques
Strings

- Could provide some leads:
  - Passwords
  - URLs
  - IP addresses (if not in binary)
  - File names and contents
- Remember to look for ASCII/ANSI and UNICODE strings!
- Expect large quantities of data and a lot of noise.
- Memory is heavily fragmented.
- Don’t jump to conclusions!

Analysis Techniques
List Walking

Enumerating the list of processes

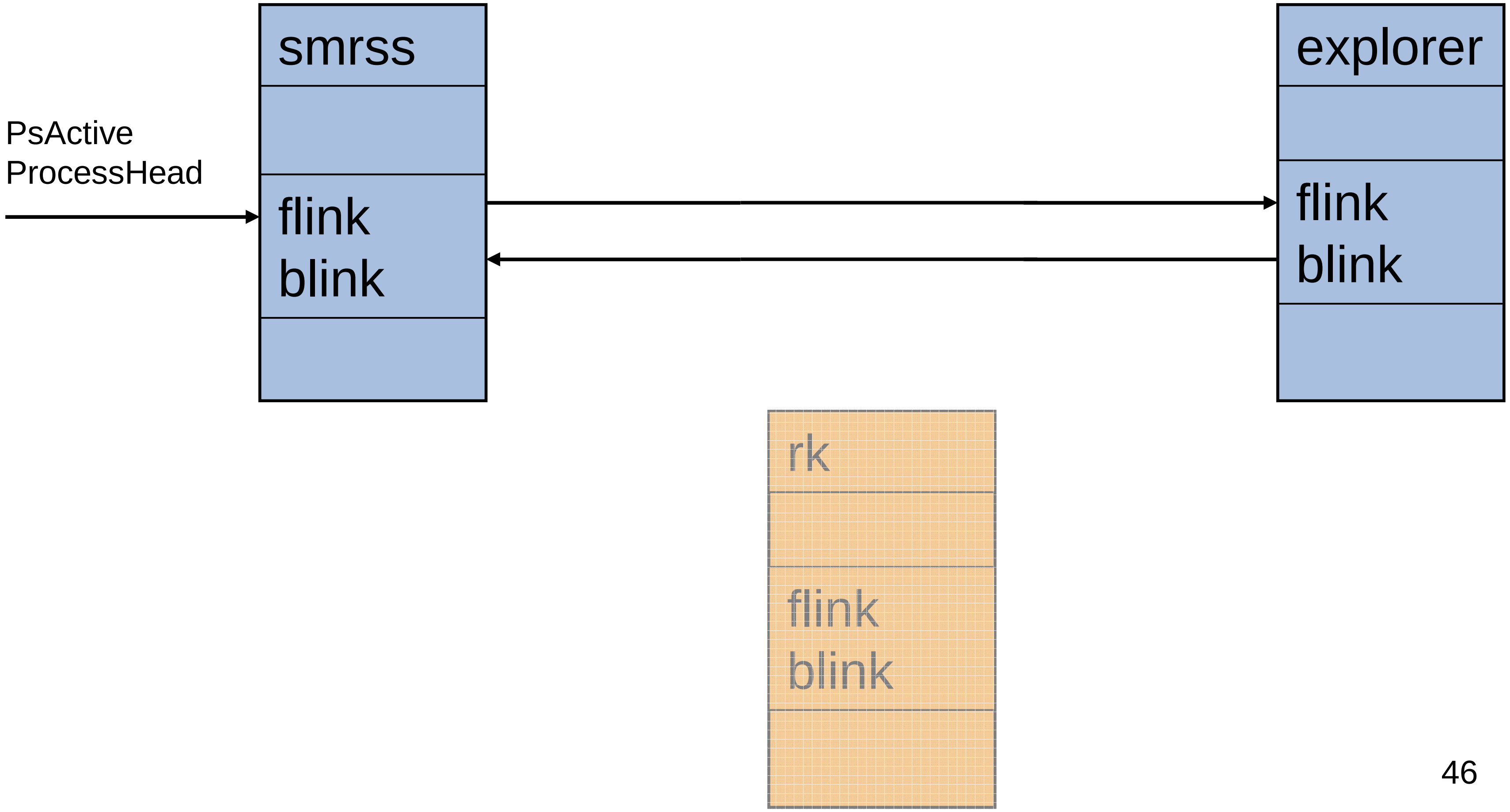


Analysis Techniques
List Walking

- Technique also applies to
  - Single lists (e.g. buckets in hash tables)
  - Trees (e.g. VAD, handles)
- Simple, fast, efficient (false positives are rare)
- Usually works well across OS version/SP/hotfix
- Possible failures:
  - OS housekeeping (e.g. terminated process, closed file)
  - non-atomic acquisition methods, broken chain
  - purposefully unlinked objects (DKOM, rootkits)

Analysis Techniques
List Walking

Anti-forensic attack: Direct Kernel Object Manipulation (DKOM)



Analysis Techniques
Scanning

- Define signature on
  - Constant parts of structure
  - Ranges of values
  - Complex conditions
- Scan whole memory image
- Slow (depending on complexity)
- Specific to OS version/SP/hotfix
- Possible failures:
  - Un-specific signature causes high rate of false positives
  - Weak signature causes false negatives (adversary modifies non-essential data to thwart detection)

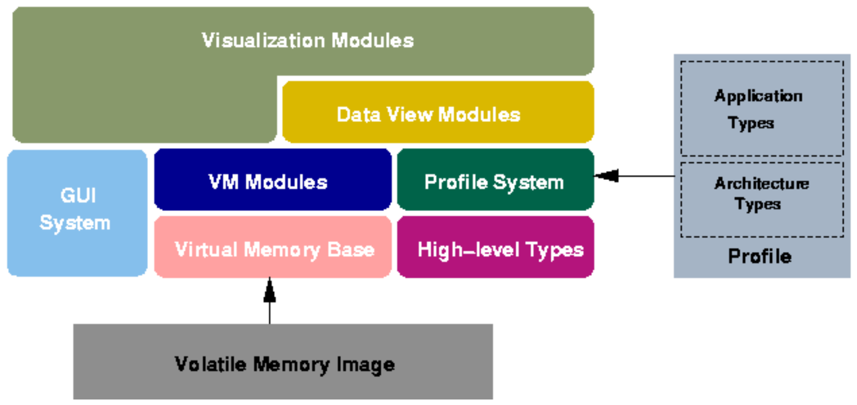
Analysis Techniques
Finding Suspicious Activity

- Cross-view detection
  - Different APIs
  - Compare results of list-waking and scanning
  - Examine any differences!
- Conformance checks
  - Null pointers
  - Iinvalid object types
  - Missing strings
  - …

Part 2
Using Volatility

Overview

- FATkit
  - Petroni and Walters, 2006
  - Layered, modular architecture
  - http://www.4tphi.net/fatkit/



- VolaTools
  - Walters and Petroni, 2007
  - Intellectual property of Komoku, sold to Microsoft in March 2008
  - Mostly open source, but closed-source address translation

- Volatility
  - Walters et al., 2007
  - Completely open source, community project
  - https://www.volatilesystems.com/

Ressources

- Mailing list
  - use of the tools and general questions
    vol-users@volatilesystems.com
  - New features and design decisions
    vol-dev@volatilesystems.com

- Chat (IRC): #volatility@freenode.net

- Blogs
  - http://volatilesystems.blogspot.com/
  - http://volatility.tumblr.com/

Overview
Contributors

- Code Contributors
  - Michael Cohen
  - David Collett
  - Brendan Dolan-Gavitt
  - Blake Matheny
  - Andreas Schuster
- Research Collaborators
  - Jide Abu
  - Jose Nazario
  - Doug White
  - Matthieu Suiche
- Testing/Bugs
  - Joseph Ayo Akinyele
  - Tommaso Assandri
  - Brian Carrier
  - Harlan Carvey
  - Eoghan Casey
  - Jim Clausing
  - Jon Evans
  - Robert Guess
  - Jesse Kornblum
  - Jamie Levy
  - Eugene Libster
  - Erik Ligda
  - Tony Martin
  - Golden G. Richard III
  - Sam F. Stover

Overview
Prerequisites

- Python 2.5
  - Windows users: Active State Python http://www.activestate.com/activepython
- Volatility
  - stable https://www.volatilesystems.com/default/volatility
  - SVN on http://code.google.com/p/volatility/, see instructions
- Plug-ins may require additional software, e.g.
  - pefile http://code.google.com/p/pefile/
  - pydasm http://dkbza.org/pydasm.html

Overview
Plug-ins

- Comprehensive, but unofficial list of Volatility plug-ins
  http://www.forensicswiki.org/wiki/List_of_Volatility_Plugins
- Standard procedure: install into `memory_plugins` subdirectory
- Some plug-ins may depend on additional python modules or require different installation procedures!
- Run `python volatility` – the new command(s) should now appear.
- Run `python volatility` *`command`* `--help` to learn about the syntax.

Commands

Commands
Getting Help

- For a list of internal- and plug-in commands:
  ```
  python volatility
  ```

- For help on any command:
  ```
  python volatility command --help
  ```

Commands
Standard Options

- `-f` *`FILENAME`*
  `--file=`*`FILENAME`*
  Path and name of memory image

- `-b` *`BASE_ADDRESS`*
  `--base=`*`BASE_ADDRESS`*
  Physical offset (in hex!) of Directory Table Base (CR3)

- `-t` *`TYPE`*
  `--type=`*`TYPE`*
  Type of memory image. Valid parameters are:
  - `auto` (default)
  - `pae`
  - `nopae`

Commands
Information about the Memory Image

- ident

```
Image Name: /samples/hxdef.dd
Image Type: Service Pack 2
VM Type: nopae
DTB: 0x39000
Datetime: Fri Apr 10 10:58:53 2009
```

- datetime

```
Image local date and time: Fri Apr 10 10:58:53 2009
```

- Both commands report the system’s local time!
- datetime on DVD has been modified to report time in UTC, too.

Commands

Hands-on: Information about the Memory Image

- Analyze memory image “/samples/exemplar13.vmem” by hogfly.
- Authenticate the memory image
  `MD5 5ec0c6dffa29b1bd5a6cbec1829df25d`
- Determine the OS version and the system’s time. This becomes the endpoint of our timeline.

Commands

Hands-on: Information about the Memory Image

- Authenticate the memory image
  `MD5 5ec0c6dffa29b1bd5a6cbec1829df25d`

```
md5sum /samples/exemplar13.vmem
5ec0c6dffa29b1bd5a6cbec1829df25d
```

Match!

Commands
Hands-on: Information about the Memory Image

- Determine the OS version and the system’s time. This will become the latest point in our timeline.

```
> python volatility ident –f /samples/exemplar13.vmem
 Image Name: /samples/exemplar13.vmem
              Image Type: Service Pack 2
                 VM Type: pae
                     DTB: 0x7d0000
                Datetime: Wed Jan 07 20:54:57 2009

> python volatility datetime –f /samples/exemplar13.vmem
Image local date and time: Wed Jan 07 20:54:57 2009
Image date and time (UTC): Thu Jan 08 01:54:57 2009
```

Commands

Hands-on: Timeline

Thu Jan 08 01:54:57 2009	**memory image obtained**

Commands
Threads

- thrdscan
  - Searches for DISPATCHER_HEADER
  - Applies several constraints
  - Based on PTFinder, though less strict constraints
  - Slow
- thrdscan2
  - Searches for POOL_HEADER
  - Applies only a few constraints
  - Fast
  - Does not detect the idle thread

Options

- thrdscan
  - `-s` *`HEXADDRESS`*
    `--start=`*`HEXADDRESS`*
    Start address

  - `-e` *`HEXADDRESS`*
    `--end=`*`HEXADDRESS`*
    End address

  - `-s`
    `--slow`
    Perform scan on original address space instead of flat file

Commands
Threads

Output format

- Number
- Unique Process ID (PID)
- Thread ID (TID)
- Physical offset into memory image

```
No.  PID    TID    Offset
---- ------ ------ ----------

   1    888   1716 0x0008a020
   2    888   1712 0x0008ada8
   3   1296   1384 0x001a5230
```

- Version on DVD also reports thread creation and exit times.

Commands
Modules

- modules
  - Starts off from PsLoadedModuleList
  - Traverses list of loaded modules (in load order)

- modscan / modscan2
  - searches for POOL_HEADER
  - modscan2 is much faster!

Options

- modscan
  - `-s` *`HEXADDRESS`*
    `--start=`*`HEXADDRESS`*
    Start address
  - `-e` *`HEXADDRESS`*
    `--end=`*`HEXADDRESS`*
    End address
  - `-s`
    `--slow`
    Perform scan on original address space instead of flat file

Commands
Modules

- Output format
  - File name
  - Base address
  - Size in bytes
  - Module name

- All three functions share a common output format!

moddump plug-in

- Written by Brendan Dolan-Gavitt
  http://moyix.blogspot.com/2008/10/plugin-post-moddump.html
- Dumps loaded kernel module(s) to disk
- Command line options
  - `-m MODE`
    `--mode=MODE`
  - `-u`
    `--unsafe`
  - `-o OFFSET`
    `--offset=OFFSET`
  - `-p REGEX`
    `--pattern=REGEX`
  - `-i`
    `--ignore-case`

Commands
Processes

- pslist
  - Starts off from PsActiveProcessHead
  - Traverses EPROCESS. ActiveProcessLinks

- psscan
  - Searches for DISPATCHER_HEADER (finds Idle process)
  - Applies several constraints
  - Based on PTFinder, though less strict
  - Slow

- psscan2
  - Searches for POOL_HEADER
  - Applies only a few constraints
  - Fast

Options

- psscan
  - `-s` *`HEXADDRESS`*
    `--start=`*`HEXADDRESS`*
    Start address
  - `-e` *`HEXADDRESS`*
    `--end=`*`HEXADDRESS`*
    End address
  - `-s`
    `--slow`
    Perform scan on original address space instead of flat file

- psscan and psscan2
  - `-d` *`FILE`*
    `--dot=`*`FILE`*
    Draw process tree in DOT format for GraphViz

Commands
Processes

- Output format (common data)
  - Name (shortened to 16 characters)
  - Unique Process ID (PID)
  - Parent Process ID (PPID)
  - Creation time

- Additional information:
  - Number
  - Thread count
  - Handle count
  - Exit time
  - Physical offset into memory image
  - CR3 (DTB, PDB, ...)

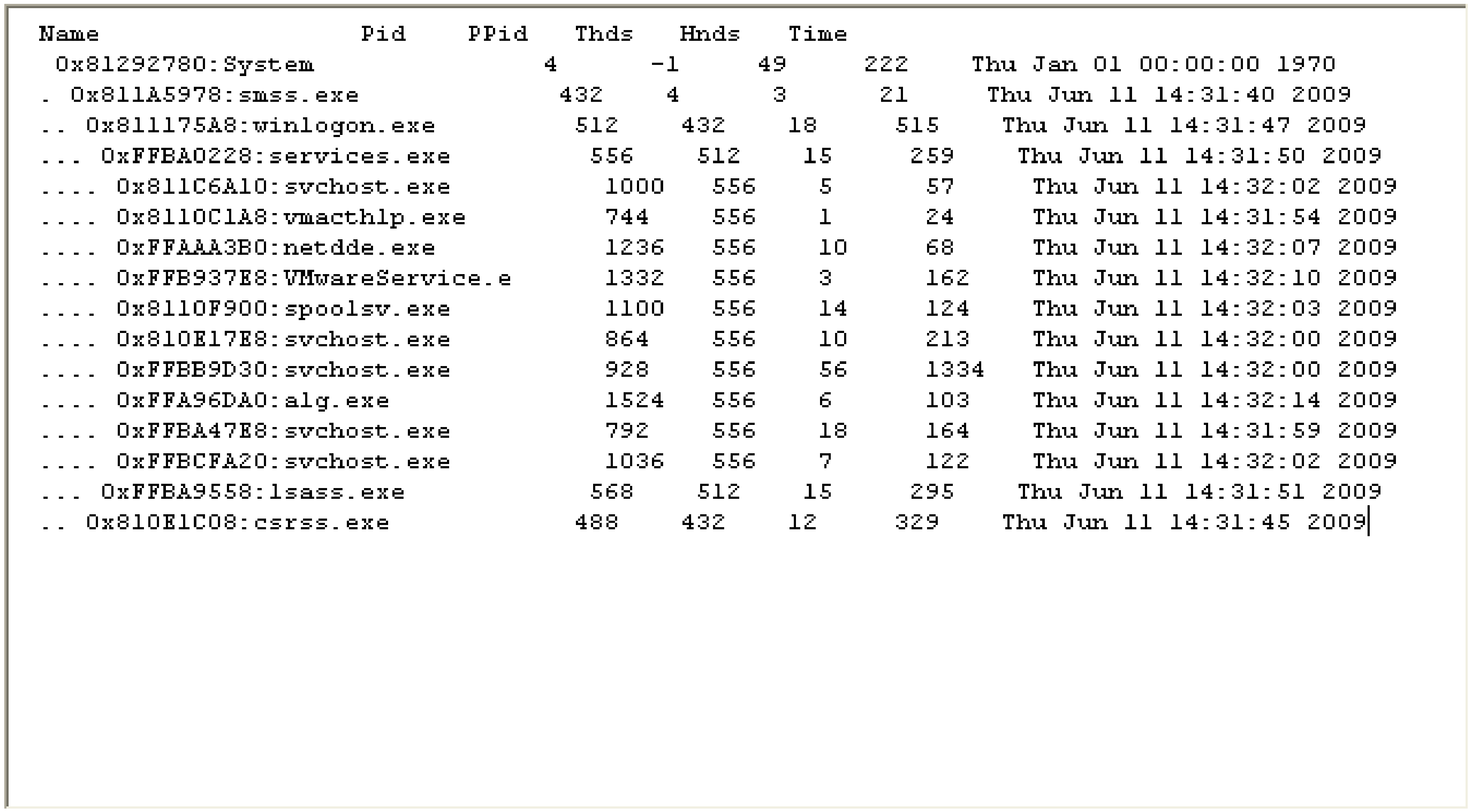
- Three functions, three different output formats!

pstree plug-in

- Written by Dr. Michael Cohen
  http://scudette.blogspot.com/2008/10/pstree-volatility-plugin.html
- Visualizes parent-child relationship through indentation
- Isolated parts of the process tree may be missing.
- `-v`
  `--verbose`
  Displays full path name (from process audit), command line and path (from process environment block PEB)

Commands

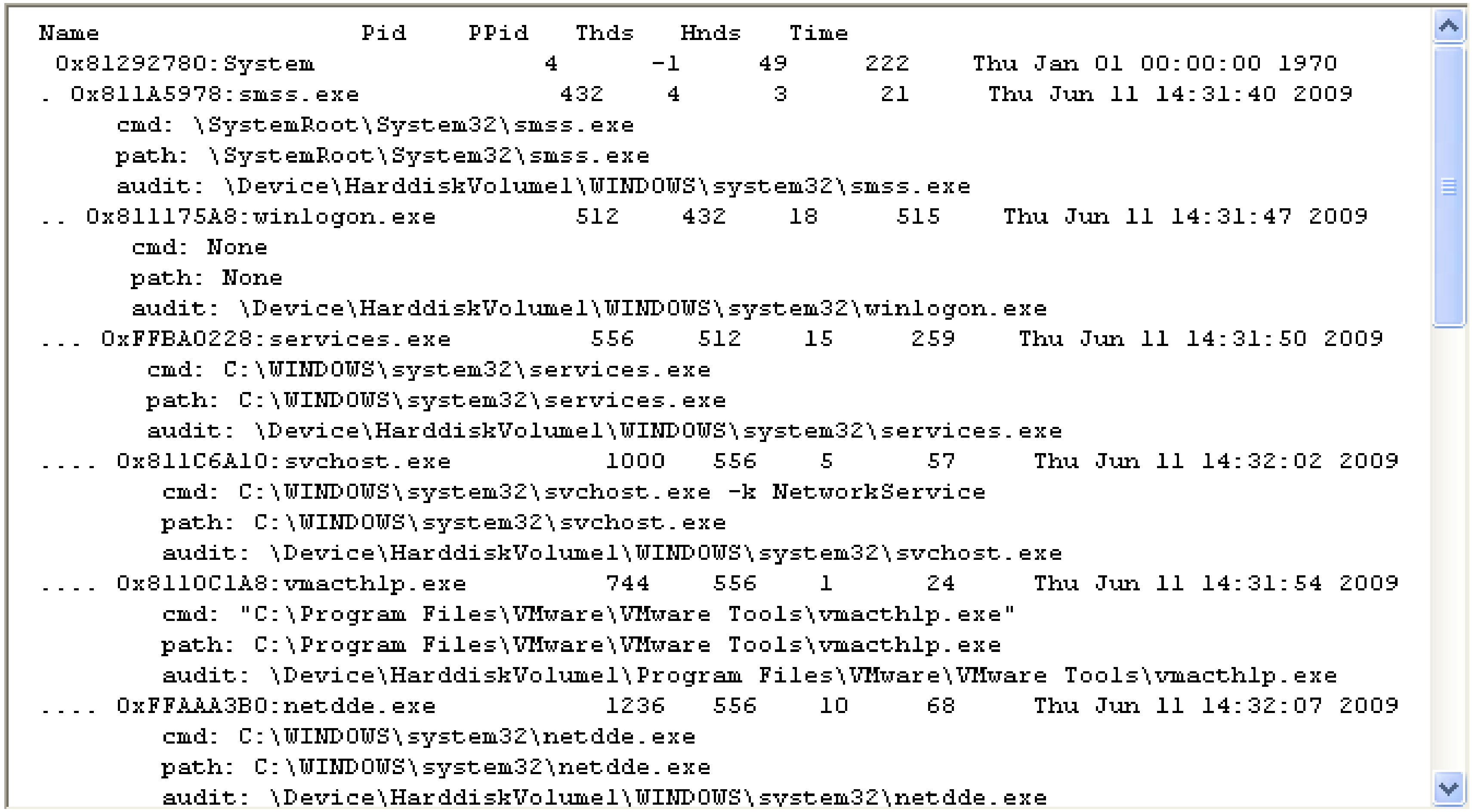
Processes



```
Name                     Pid     PPid    Thds    Hnds    Time
 0x81292780:System              4       -1      49      222     Thu Jan 01 00:00:00 1970
. 0x811A5978:smss.exe            432     4       3       21      Thu Jun 11 14:31:40 2009
.. 0x811175A8:winlogon.exe        512     432     18      515     Thu Jun 11 14:31:47 2009
... 0xFFBA0228:services.exe        556     512     15      259     Thu Jun 11 14:31:50 2009
.... 0x811C6A10:svchost.exe         1000    556     5       57      Thu Jun 11 14:32:02 2009
.... 0x8110C1A8:vmacthlp.exe        744     556     1       24      Thu Jun 11 14:31:54 2009
.... 0xFFAAA3B0:netdde.exe          1236    556     10      68      Thu Jun 11 14:32:07 2009
.... 0xFFB937E8:VMwareService.e     1332    556     3       162     Thu Jun 11 14:32:10 2009
.... 0x8110F900:spoolsv.exe         1100    556     14      124     Thu Jun 11 14:32:03 2009
.... 0x810E17E8:svchost.exe         864     556     10      213     Thu Jun 11 14:32:00 2009
.... 0xFFBB9D30:svchost.exe         928     556     56      1334    Thu Jun 11 14:32:00 2009
.... 0xFFA96DA0:alg.exe             1524    556     6       103     Thu Jun 11 14:32:14 2009
.... 0xFFBA47E8:svchost.exe         792     556     18      164     Thu Jun 11 14:31:59 2009
.... 0xFFBCFA20:svchost.exe         1036    556     7       122     Thu Jun 11 14:32:02 2009
... 0xFFBA9558:lsass.exe           568     512     15      295     Thu Jun 11 14:31:51 2009
.. 0x810E1C08:csrss.exe           488     432     12      329     Thu Jun 11 14:31:45 2009
```

Commands

Processes



```
Name                     Pid     PPid    Thds    Hnds    Time
 0x81292780:System                  4       -1      49      222     Thu Jan 01 00:00:00 1970
. 0x811A5978:smss.exe                432     4       3       21      Thu Jun 11 14:31:40 2009
     cmd: \SystemRoot\System32\smss.exe
     path: \SystemRoot\System32\smss.exe
     audit: \Device\HarddiskVolume1\WINDOWS\system32\smss.exe
.. 0x811175A8:winlogon.exe            512     432     18      515     Thu Jun 11 14:31:47 2009
      cmd: None
      path: None
      audit: \Device\HarddiskVolume1\WINDOWS\system32\winlogon.exe
... 0xFFBA0228:services.exe            556     512     15      259     Thu Jun 11 14:31:50 2009
       cmd: C:\WINDOWS\system32\services.exe
       path: C:\WINDOWS\system32\services.exe
       audit: \Device\HarddiskVolume1\WINDOWS\system32\services.exe
.... 0x811C6A10:svchost.exe            1000    556     5       57      Thu Jun 11 14:32:02 2009
        cmd: C:\WINDOWS\system32\svchost.exe -k NetworkService
        path: C:\WINDOWS\system32\svchost.exe
        audit: \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
.... 0x8110C1A8:vmacthlp.exe           744     556     1       24      Thu Jun 11 14:31:54 2009
        cmd: "C:\Program Files\VMware\VMware Tools\vmacthlp.exe"
        path: C:\Program Files\VMware\VMware Tools\vmacthlp.exe
        audit: \Device\HarddiskVolume1\Program Files\VMware\VMware Tools\vmacthlp.exe
.... 0xFFAAA3B0:netdde.exe             1236    556     10      68      Thu Jun 11 14:32:07 2009
        cmd: C:\WINDOWS\system32\netdde.exe
        path: C:\WINDOWS\system32\netdde.exe
        audit: \Device\HarddiskVolume1\WINDOWS\system32\netdde.exe
```

Commands
Hands-on: Processes

- Analyze memory image “/samples/exemplar13.vmem” by hogfly.
- Find the PID, start/end times and exit code for processes
  - explorer.exe
  - ud32.exe

Commands
Hands-on: Timeline

Thu Jan 08 01:53:09 2009	**processes 464 and 1040 (ud32.exe) started by process 1928 (explorer.exe)**
Thu Jan 08 01:53:10 2009	**process 1040 terminated, exit code 0**
Thu Jan 08 01:54:57 2009	memory image obtained

Commands
Per-Process Information

dlllist

- Enumerates DLLs (and EXEs) loaded by a process
- Does not work for terminated or hidden processes
- `-p` *`PID`*
  `--pid=`*`PID`*

```
explorer.exe pid: 2032
Command line : C:\WINDOWS\Explorer.EXE
Service Pack 2

Base         Size         Path
0x1000000    0xff000      C:\WINDOWS\Explorer.EXE
0x7c900000   0xb0000      C:\WINDOWS\system32\ntdll.dll
0x7c800000   0xf4000      C:\WINDOWS\system32\kernel32.dll
```

Commands
Per-Process Information

files

- Enumerates file handles that were opened by a process

- `-p` *`PID`*
  `--pid=`*`PID`*

```
Pid: 2032
File    \Documents and Settings\All Users\Desktop
File    \Documents and Settings\TestUser\Desktop
File    \Documents and Settings\TestUser\Start Menu
File    \Documents and Settings\TestUsers\Start Menu
File    \wkssvc
```

Commands
Per-Process Information

getsids plug-in

- Written by Grendan Dolan-Gavitt
  http://moyix.blogspot.com/2008/08/linking-processes-to-users.html

- Does not examine terminated and hidden processes

```
VMwareService.e (1332): S-1-5-18 (Local System)
VMwareService.e (1332): S-1-5-32-544 (Administrators)
VMwareService.e (1332): S-1-1-0 (Everyone)
VMwareService.e (1332): S-1-5-11 (Authenticated Users)
alg.exe (1524): S-1-5-19 (NT Authority)
alg.exe (1524): S-1-1-0 (Everyone)
alg.exe (1524): S-1-5-32-545 (Users)
alg.exe (1524): S-1-5-6 (Service)
```

Commands
Per-Process Information

memmap

- Displays mapping between virtual and physical addresses

memdmp

- Dumps process memory
- Command line options
  - `-o HEXOFFSET`
    `--offset=HEXOFFSET`
  - `-p` *`PID`*
    `--pid=`*`PID`*

Commands
Per-Process Information

procdump

- Dumps the executable into a file
- The executable is likely to crash (state!)
- Great command for static analysis, though

- Command line options
  - `-o HEXOFFSET`
    `--offset=HEXOFFSET`
  - `-p` *`PID`*
    `--pid=`*`PID`*

Commands
Network Sockets

- sockets
  - Locates tcpip module
  - Looks for list head at known offsets into module
  - Traverses list of socket objects

- sockscan / sockscan2
  - Searches for POOL_HEADER
  - sockscan2 is much faster!

Options

- sockscan
  - `-s` *`HEXADDRESS`*
    `--start=`*`HEXADDRESS`*
    Start address

  - `-e` *`HEXADDRESS`*
    `--end=`*`HEXADDRESS`*
    End address

  - `-s`
    `--slow`
    Perform scan on original address space instead of flat file

Commands
Network Sockets

- Output format
  - Unique Process ID (PID)
  - Port (if applicable)
  - Protocol
  - Create time

- Output formats differ slightly.

Commands
Network Sockets

- sockets

```
Pid      Port     Proto    Create Time
4        1026     6        Thu Jun 11 14:32:15 2009
4        0        47       Thu Jun 11 14:32:15 2009
928      0        2        Thu Jun 11 14:32:13 2009
4        445      6        Thu Jun 11 14:31:28 2009
```

- sockscan / sockscan2

```
PID      Port     Proto    Create Time                  Offset
------ ------ ------ -------------------------- ----------

1524     1025     6        Thu Jun 11 14:32:15 2009     0x0083c838
4        1026     6        Thu Jun 11 14:32:15 2009     0x01031620
1640     31337    6        Thu Jun 11 14:35:15 2009     0x0104eb78
4        138      17       Thu Jun 11 14:32:06 2009     0x01057e98
```

Commands
Network Connections

- connections
  - Locates tcpip module
  - Looks for TCBtable at known offsets into module
  - Locates and dumps connection objects

- connscan / connscan2
  - Searches for POOL_HEADER
  - connscan2 is much faster!

Commands
Network Connections

Options

- connscan
  - `-s` *`HEXADDRESS`*
    `--start=`*`HEXADDRESS`*
    Start address
  - `-e` *`HEXADDRESS`*
    `--end=`*`HEXADDRESS`*
    End address
  - `-s`
    `--slow`
    Performs scan on original address space instead of flat file

Commands
Network Connections

- Output format
  - Local IP address and port
  - Remote IP address and port
  - Unique Process ID (PID)

- Output formats differ slightly.

Commands
Network Connections

- connections

```
Local Address             Remote Address            Pid
192.168.242.128:135       192.168.242.1:1777        848
```

- connscan / connscan2

```
Local Address             Remote Address            Pid
------------------------- ------------------------- ------

192.168.242.128:135       192.168.242.1:1777        848
```

Commands
Hands-on: Processes

- Analyze memory image “/samples/exemplar13.vmem” by hogfly.
- Find network sockets and connections opened by the following processes
  - explorer.exe (PID 1928)
  - ud32.exe (PID 464 and 1040)

Commands

Hands-on: Timeline

Thu Jan 08 01:53:07 2009	**process 1928 (explorer.exe) creates socket for port 1048/tcp, connects to 67.215.11.138:7000**
Thu Jan 08 01:53:09 2009	**process 1928 (explorer.exe) creates sockets for ports 1049/tcp and 1050/tcp, and connects both to 72.10.166.195:80** processes 464 and 1040 (ud32.exe) started by process 1928 (explorer.exe)
Thu Jan 08 01:53:10 2009	**process 464 creates sockets for ports 27714/tcp and 1052/udp** process 1040 terminated, exit code 0
Thu Jan 08 01:54:57 2009	memory image obtained

Commands
Registry

regobjkeys

- Lists opened registry keys
- Command line options
  - `-o HEXOFFSET`
    `--offset=HEXOFFSET`
  - `-p` *`PID`*
    `--pid=`*`PID`*

```
Pid: 464
\REGISTRY\MACHINE
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPIP\PARAMETERS
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\NETBT\PARAMETERS
\REGISTRY\USER\S-1-5-21-1614895754-1604221776-839522115-
  1003\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\INTERNET SETTINGS
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETER
  S\PROTOCOL_CATALOG9
```

VolReg plug-in package

- Written by Brendan Dolan-Gavitt
  http://moyix.blogspot.com/2009/06/volreg-06-now-with-bigdata.html

- Installation
  - Some modules depend on PyCrypto
    http://www.amk.ca/python/code/crypto.html
  - Windows binary distribution at
    http://www.voidspace.org.uk/python/modules.shtml

VolReg plug-in package

- Preparation
  - call hivescan to scan for _CMHIVE structures
  - call hivelist on any of the found structures to map them to hive files

- Data access
  - hivedump
    - →dumps whole hives (optional: with values)
    - →timestamps in local time zone of the analysis workstation
  - printkey
    - →queries a single key
    - →timestamps in local time zone of the analysis workstation
    - →do not escape backslash on Windows!

Commands
Hands-on: Registry

- Analyze the memory image “exemplar13.vmem” by hogfly.

- Examine some well-known autostart entries:
  - HKCU\Software\Microsoft\Windows\CurrentVersion\Run
  - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows
  - HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

- A comprehensive list of launch and hijack points can be found at http://www.silentrunners.org/sr_launchpoints.html

- Create a timeline of events for the whole registry.

Commands

Hands-on: Registry

```
python volatility hivescan -f /samples/exemplar13.vmem
Offset          (hex)
34786144        0x212cb60
35029896        0x2168388
36798472        0x2318008
52190048        0x31c5b60
61227776        0x3a64300
62263304        0x3b61008
62692192        0x3bc9b60
78032904        0x4a6b008
117499936       0x700e820
117721952       0x7044b60
118016032       0x708c820
181174280       0xacc8008
182220832       0xadc7820
```

Commands

Hands-on: Registry

```
python volatility hivelist -f /samples/exemplar13.vmem
 -o 0x212cb60
Address      Name
0xe179e008   [no name]
0xe1a58b60   \Documents and Settings\foo\NTUSER.DAT
0xe1548008   [no name]
0xe1535820   \Documents and Settings\LocalService\NTUSER.DAT
0xe1095820   [no name]
0xe107e820   \Documents and Settings\NetworkService\NTUSER.DAT
0xe13a3008   \WINDOWS\system32\config\software
0xe1397300   \WINDOWS\system32\config\default
0xe13a0b60   \WINDOWS\system32\config\SECURITY
0xe1362b60   \WINDOWS\system32\config\SAM
0xe11c2008   [no name]
0xe1018388   \WINDOWS\system32\config\system
0xe1008b60   [no name]
```

Commands
Hands-on: Registry

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run

```
Address       Name
0xe1a58b60    \Documents and Settings\foo\NTUSER.DAT

> python volatility printkey –f /samples/exemplar13.vmem
  -o 0xe1a58b60 'Software\Microsoft\Windows\CurrentVersion\Run'

'Software\Microsoft\Windows\CurrentVersion\Run'
Key name: Run (Stable)
Last updated: Thu Jan 08 01:53:10 2009

Subkeys:

Values:
REG_SZ    Windows Network Data Management System Service :
  "ud32.exe" *  (Stable)
```

Commands
Hands-on: Registry

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows

```
Address       Name
0xe13a3008    \WINDOWS\system32\config\software

> python volatility printkey –f /samples/exemplar13.vmem
  -o 0xe13a3008 'Microsoft\Windows NT\CurrentVersion\Windows'

'Microsoft\Windows NT\CurrentVersion\Windows'
Key name: Windows (Stable)
Last updated: Thu Jan 08 01:53:10 2009

Subkeys:

Values:
REG_SZ    AppInit_DLLs :    (Stable)
REG_SZ    Spooler      : yes  (Stable)
REG_SZ    load         : ud32.exe  (Stable)
```

Commands

Hands-on: Registry

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

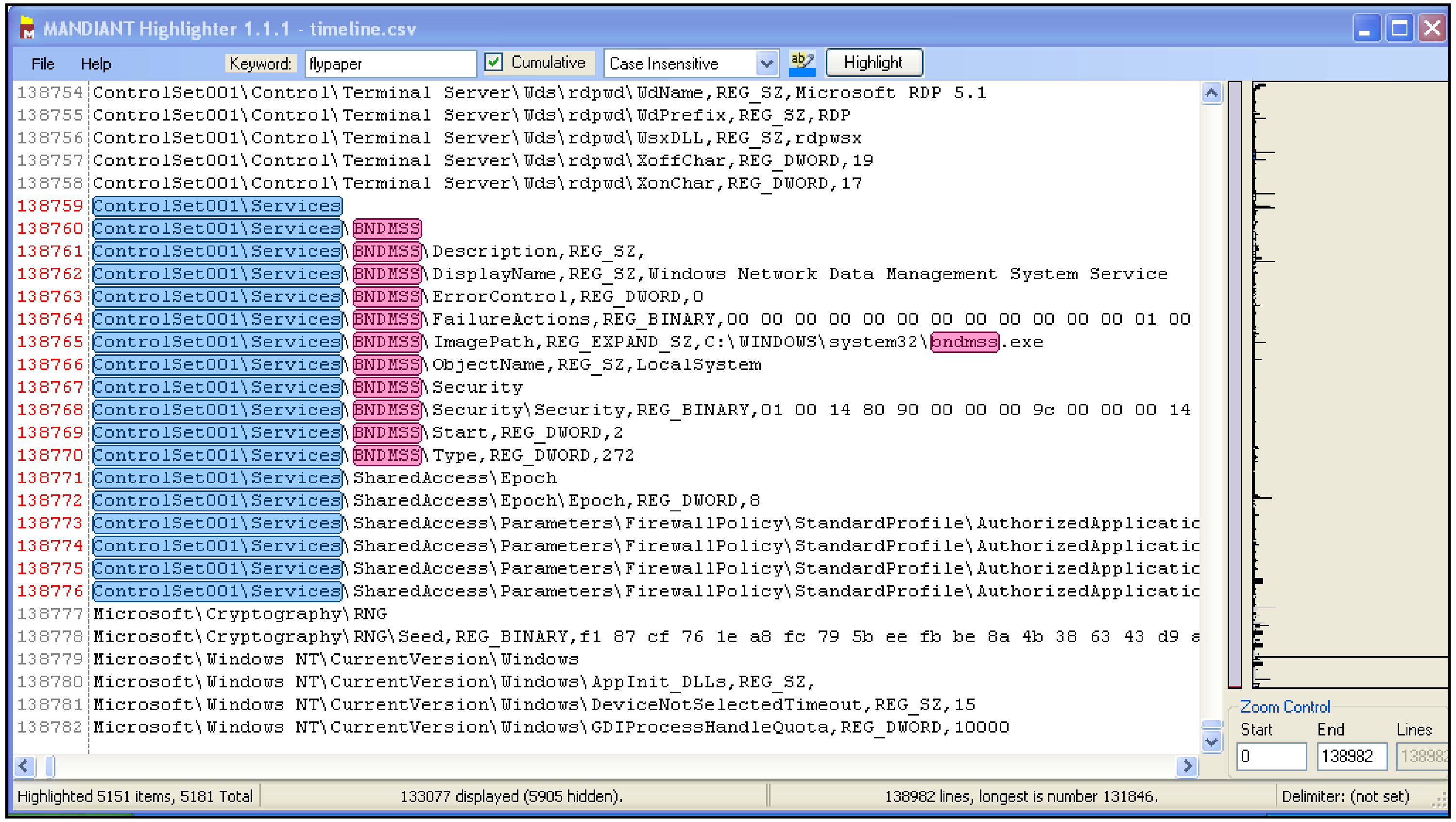
```
"Microsoft\Windows NT\CurrentVersion\Winlogon"
Key name: Winlogon (Stable)
Last updated: Thu Jan 08 01:53:10 2009

Subkeys:
   GPExtensions (Stable)
   Notify (Stable)
   SpecialAccounts (Stable)
   Credentials (Volatile)

Values:
REG_SZ    DefaultDomainName : EXEMPLARXP  (Stable)
REG_SZ    DefaultUserName : foo  (Stable)
REG_SZ    Shell         : Explorer.exe  (Stable)
REG_SZ    Userinit      :
  C:\WINDOWS\system32\userinit.exe,ud32.exe  (Stable)
```

Commands
Hands-on: Registry

- Create a timeline of events for the whole registry.

```
> python volatility hivedump -f /samples/exemplar13.vmem -o 0x212cb60 -v
Dumping  => e179e008.csv
Dumping \Documents and Settings\foo\NTUSER.DAT => e1a58b60.csv
Dumping  => e1548008.csv
Dumping \Documents and Settings\LocalService\NTUSER.DAT => e1535820.csv
Dumping  => e1095820.csv
Dumping \Documents and Settings\NetworkService\NTUSER.DAT => e107e820.csv
Dumping \WINDOWS\system32\config\software => e13a3008.csv
Dumping \WINDOWS\system32\config\default => e1397300.csv
Dumping \WINDOWS\system32\config\SECURITY => e13a0b60.csv
Dumping \WINDOWS\system32\config\SAM => e1362b60.csv
Dumping  => e11c2008.csv
Dumping \WINDOWS\system32\config\system => e1018388.csv
Dumping  => e1008b60.csv

> sort –n *.csv > timeline.csv
```

Commands
Hands-on: Registry



```
ControlSet001\Control\Terminal Server\Wds\rdpwd\WdName,REG_SZ,Microsoft RDP 5.1
ControlSet001\Control\Terminal Server\Wds\rdpwd\WdPrefix,REG_SZ,RDP
ControlSet001\Control\Terminal Server\Wds\rdpwd\WsxDLL,REG_SZ,rdpwsx
ControlSet001\Control\Terminal Server\Wds\rdpwd\XoffChar,REG_DWORD,19
ControlSet001\Control\Terminal Server\Wds\rdpwd\XonChar,REG_DWORD,17
ControlSet001\Services
ControlSet001\Services\BNDMSS
ControlSet001\Services\BNDMSS\Description,REG_SZ,
ControlSet001\Services\BNDMSS\DisplayName,REG_SZ,Windows Network Data Management System Service
ControlSet001\Services\BNDMSS\ErrorControl,REG_DWORD,0
ControlSet001\Services\BNDMSS\FailureActions,REG_BINARY,00 00 00 00 00 00 00 00 00 00 00 00 01 00
ControlSet001\Services\BNDMSS\ImagePath,REG_EXPAND_SZ,C:\WINDOWS\system32\bndmss.exe
ControlSet001\Services\BNDMSS\ObjectName,REG_SZ,LocalSystem
ControlSet001\Services\BNDMSS\Security
ControlSet001\Services\BNDMSS\Security\Security,REG_BINARY,01 00 14 80 90 00 00 00 9c 00 00 00 14
ControlSet001\Services\BNDMSS\Start,REG_DWORD,2
ControlSet001\Services\BNDMSS\Type,REG_DWORD,272
ControlSet001\Services\SharedAccess\Epoch
ControlSet001\Services\SharedAccess\Epoch\Epoch,REG_DWORD,8
ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplicatic
ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplicatic
ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplicatic
ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplicatic
Microsoft\Cryptography\RNG
Microsoft\Cryptography\RNG\Seed,REG_BINARY,f1 87 cf 76 1e a8 fc 79 5b ee fb be 8a 4b 38 63 43 d9 a
Microsoft\Windows NT\CurrentVersion\Windows
Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs,REG_SZ,
Microsoft\Windows NT\CurrentVersion\Windows\DeviceNotSelectedTimeout,REG_SZ,15
Microsoft\Windows NT\CurrentVersion\Windows\GDIProcessHandleQuota,REG_DWORD,10000
```

Highlighted 5151 items, 5181 Total | 133077 displayed (5905 hidden). | 138982 lines, longest is number 131846. | Delimiter: (not set)

MANDIANT Highligher

http://www.mandiant.com/software/highlighter.htm

Commands
Hands-on: Timeline

Thu Jan 08 01:52:50 2009	**http://192.168.30.129/malware/sys32.exe executed** **sys32.exe and flypaper.exe saved to foo’s desktop**
Thu Jan 08 01:53:07 2009	process 1928 (explorer.exe) creates socket for port 1048/tcp, connects to 67.215.11.138:7000 **sys32.exe entry for Active Setup**
Thu Jan 08 01:53:09 2009	process 1928 (explorer.exe) creates sockets for ports 1049/tcp and 1050/tcp, and connects both to 72.10.166.195:80 processes 464 and 1040 (both are instances of ud32.exe) started by process 1928 (explorer.exe)
Thu Jan 08 01:53:10 2009	process 464 creates sockets for ports 27714/tcp and 1052/udp process 1040 terminated, exit code 0 **service “BNDMSS” created/modified** **firewall opened for BNDMSS and ud32.exe**
Thu Jan 08 01:54:57 2009	memory image obtained

Commands
More Kernel Objects

- Plug-ins by Andreas Schuster
  http://computer.forensikblog.de/files/volatility_plugins/

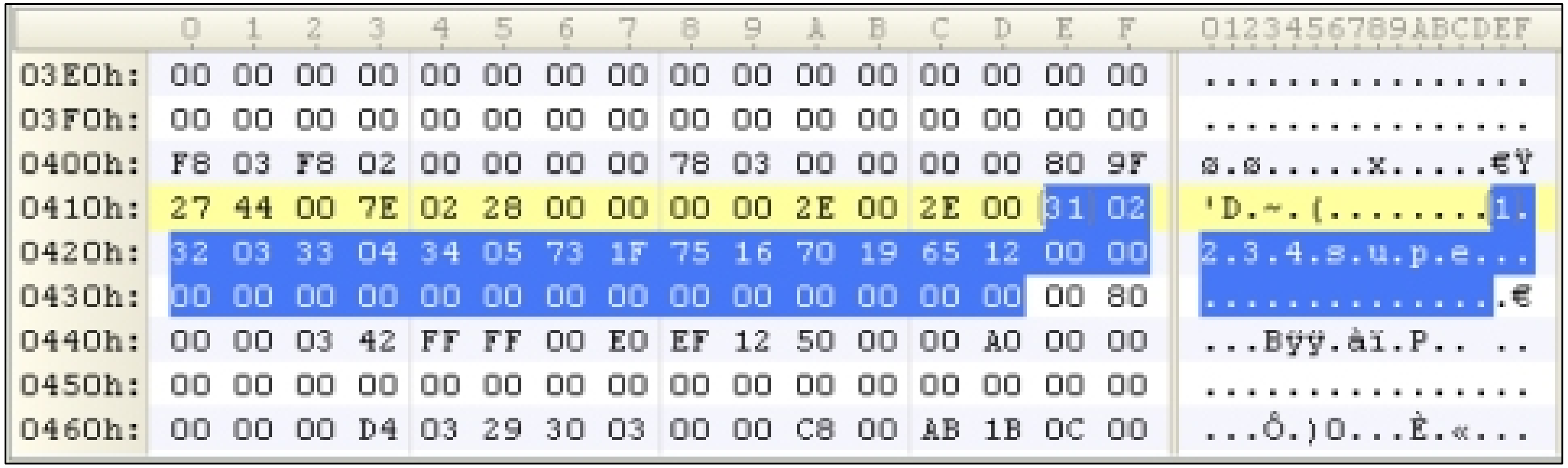
  - objtypescan - Scans for object type objects
  - driverscan - Scans for driver objects
  - fileobjscan - Scans for file objects and displays the owner
  - jobscan - Scans for job objects and their processes
  - mutantscan - Scans for mutants (mutexes)
  - symlinkobjscan - Scans for symbolic links

Commands
Secrets

- cryptoscan
  - by Jesse Kornblum
    http://jessekornblum.com/tools/volatility/cryptoscan.py
  - finds TrueCrypt passphrases
- suspicious
  - by Jesse Kernblum
    http://jessekornblum.com/tools/volatility/suspicious.py
  - searches for suspicious command line parameters

Commands
Secrets

- keyboardbuffer
  - by Andreas Schuster
    http://computer.forensikblog.de/files/volatility_plugins/keyboardbuffer.py
  - Builds on research by Jonathan Brossard
  - Relies on page 0 to be present in the memory image
  - Depends on hardware/software
  - Don’t expect too much from it!



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
03E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
03F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0400h: F8 03 F8 02 00 00 00 00 78 03 00 00 00 00 80 9F   ø.ø.....x.....€Ÿ
0410h: 27 44 00 7E 02 28 00 00 00 00 2E 00 2E 00 31 02   'D.~.(........1.
0420h: 32 03 33 04 34 05 73 1F 75 16 70 19 65 12 00 00   2.3.4.s.u.p.e...
0430h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80   ...............€
0440h: 00 00 03 42 FF FF 00 E0 EF 12 50 00 00 A0 00 00   ...Bÿÿ.àï.P.. ..
0450h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0460h: 00 00 00 D4 03 29 30 03 00 00 C8 00 AB 1B 0C 00   ...Ô.)0...È.«...
```

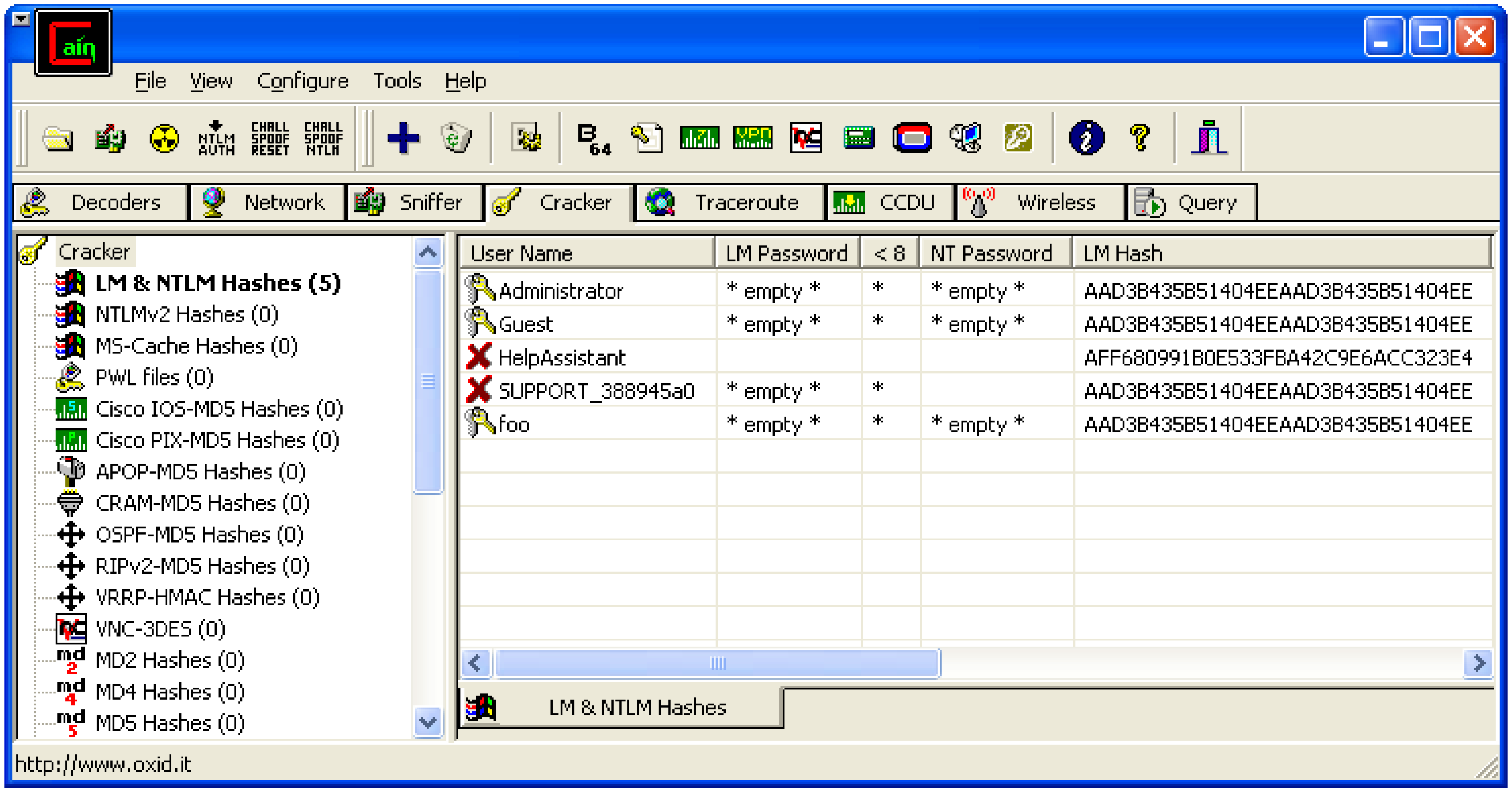
Commands
Secrets

- Part of VolReg package
  by Brendan Dolan-Gavitt
  - cachedump - Dumps cached domain credentials
  - hashdump - outputs LM/NTLM hashes in pwdump format
  - lsadump - decrypts and dumps SECURITY\Policy\Secrets

Commands
Hands-on: Registry

- Analyze the memory image “exemplar13.vmem” by hogfly.
- Dump the LM/NTLM hashes and examine their quality

Commands
Hands-on: Secrets



User Name	LM Password	< 8	NT Password	LM Hash
Administrator	* empty *	*	* empty *	AAD3B435B51404EEAAD3B435B51404EE
Guest	* empty *	*	* empty *	AAD3B435B51404EEAAD3B435B51404EE
HelpAssistant				AFF680991B0E533FBA42C9E6ACC323E4
SUPPORT_388945a0	* empty *	*		AAD3B435B51404EEAAD3B435B51404EE
foo	* empty *	*	* empty *	AAD3B435B51404EEAAD3B435B51404EE

Commands
Malware

- malfind
  - by Michael Hale Ligh
    http://mhl-malware-scripts.googlecode.com/files/malfind.py
  - Looks for (possibly) injected code
  - Invoke from Volatility base directory only!
- usermode_hooks
  - by Michael Hale Ligh
    http://mhl-malware-scripts.googlecode.com/files/usermode_hooks.py
  - Detects IAT and EAT hooks, detours
  - Depends on pydasm and pefile

Commands
Malware

- ssdt
  - by Brendan Dolan-Gavitt
    http://moyix.blogspot.com/2008/08/auditing-system-call-table.html
  - Examines System Service Descriptor Table per thread
  - You may want to filter out ntoskrnl.exe and win32k.sys

```
> python volatility ssdt -f /samples/exemplar15.vmem" |
  grep -v ntoskrnl.exe | grep -v win32k.sys

Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 80501030 with 284 entries
  Entry 0x00ad: 0xf8dfe23e (NtQuerySystemInformation) owned
  by PCIDump.SYS
SSDT[1] at bf997600 with 667 entries
```

Commands
Virtual/Physical Conversions

- memmap
  - Maps virtual to physical addresses
- strings
  - Maps a string (physical address) to process and virtual address
  - Generate table of strings using `strings –o` or a similar command
  - Edit to reduce clutter and speed up things (lookup is slow!)

Commands
Dump Format Conversions

- dmp2raw
  - Converts a crash dump into a raw memory image
- raw2dmp
  - Converts raw dump into crash dump
  - Needs to reconstruct parts of the dump header
- hibinfo
  - converts hiberfil.sys into raw dump

Part 3
Programming Volatility

Architecture

Architecture
Main Components

1. Address spaces
   - access to different memory dump formats
   - Virtual to physical address conversion
2. Profiles and objects
   - collection of data structures for different operating systems and versions
   - simplified access to structure members
3. Data view modules
   - locate, interpret and present data

Address Spaces
Overview

Purpose

- simulate random access to linear data, like in a raw/dd memory dump
  - non-contiguous files: crash dump (DMP)
  - compressed files: hibernation file
  - structured files: AFF, EWF
- translate between physical and virtual address spaces
- filter data
  - privacy preserving address space proposed by A. Walters
- provide layered abstraction of data

Address Spaces
Layers (v1.3.1)

File layer

- FileAddressSpace
- WindowsCrashDumpSpace32
- WindowsHiberFileSpace32

Virtual address layer

- IA32PagedMemory
- IA32PagedMemoryPae

Address Spaces
Class Hierarchy (SVN)

- BaseAddressSpace
    - FileAddressSpace
        - BufferAddressSpace
        - EWFAddressSpace
        - WindowsCrashDumpSpace32
        - WindowsHiberFileSpace32
    - IA32PagedMemory
        - IA32PagedMemoryPae

Address Spaces
Interface (SVN)

Common functions

- `__init__(self, base, opts)`
- `read(self, addr, len)`
- `get_available_addresses(self)`
- `is_valid_address(self, addr)`

Improved data access

- `read_long(self, addr)`
- `zread(self, vaddr, length)`

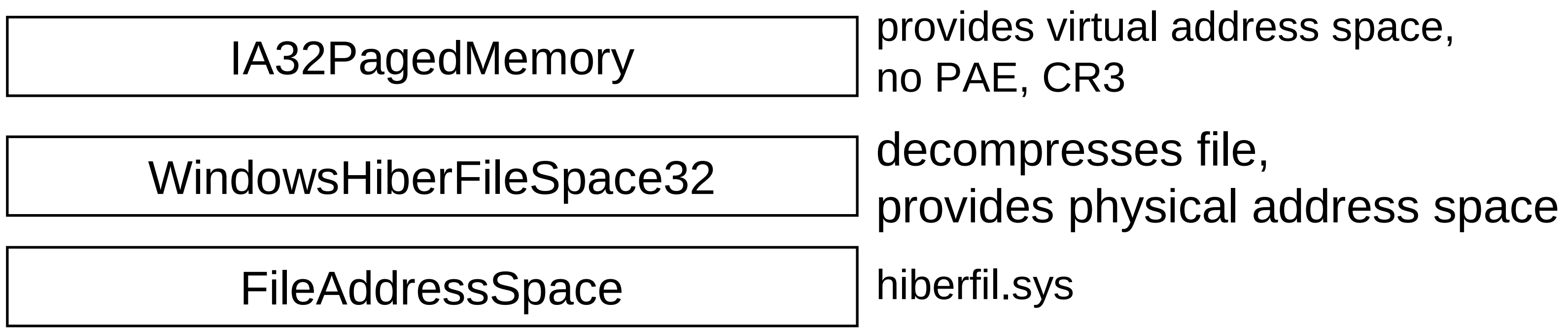
Address conversion

- `vtop(self, vaddr)`

Address Spaces
Overview

How do you access data

- in the virtual address space indicated by CR3
- in non-PAE mode
- that has been stored in hiberfil.sys?


IA32PagedMemory
provides virtual address space, no PAE, CR3
WindowsHiberFileSpace32
decompresses file, provides physical address space
FileAddressSpace
hiberfil.sys

Profiles and Objects
Overview

Purpose

- Profiles provide knowledge about
  - native types (endianess, size)
  - data structures
  - symbols (i.e. named addresses)
- Objects
  - dynamic getters for simplified data access
  - encapsulation of standard functionality,
    e.g. a process automatically providing its virtual address space

Extending Profiles
Helpful software

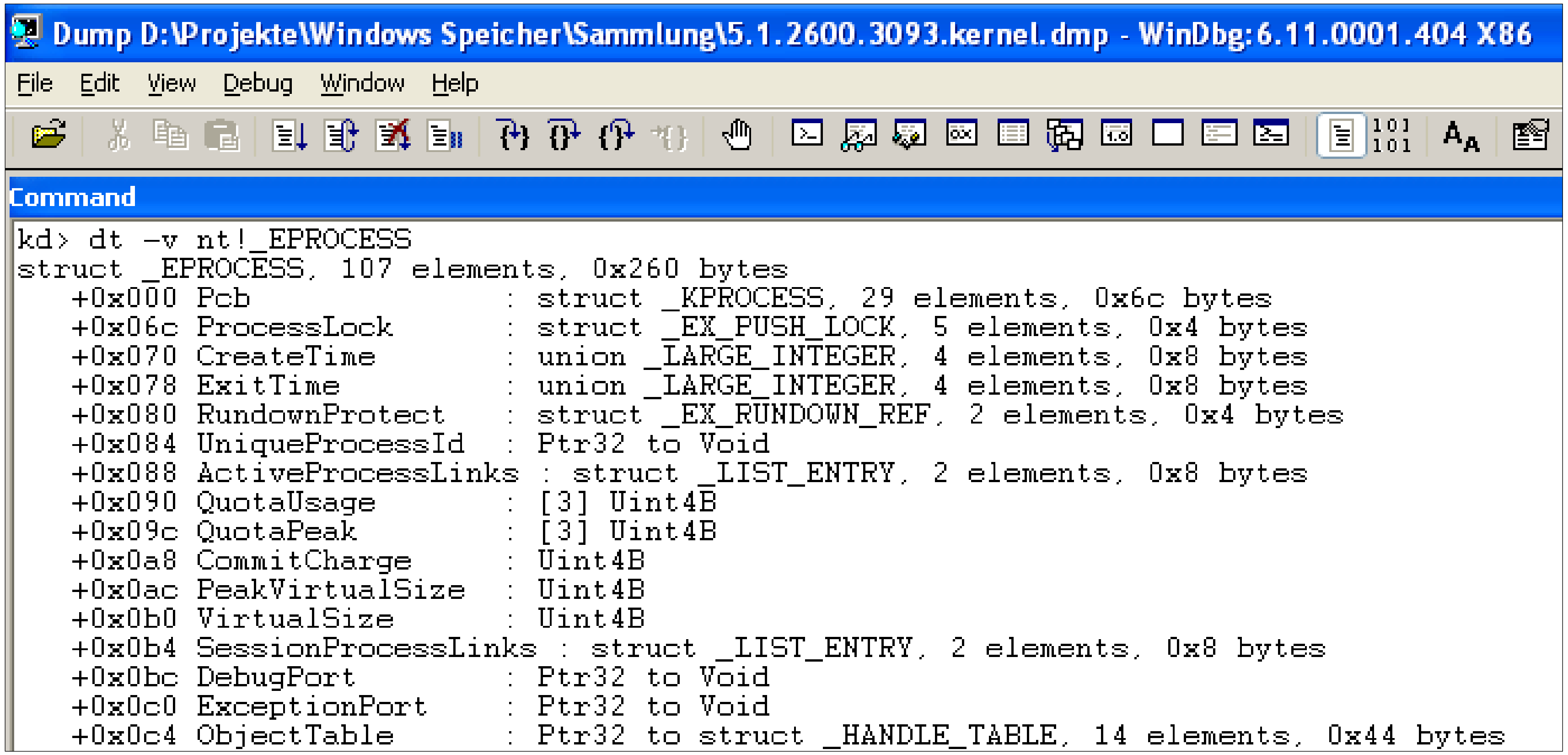
Dump debug symbols (PDB)

- Microsoft Debugger
  http://www.microsoft.com/whdc/devtools/debugging/default.mspx
- Symbol Type Viewer by Lionel d'Hauenens
  http://www.labo-asso.com/download/SymbolTypeViewer_v1.0_beta.zip
- TypeInfoDump by Oleg Starodumov:
  http://www.debuginfo.com/tools/typeinfodump.html

Reverse-engineer kernel and drivers

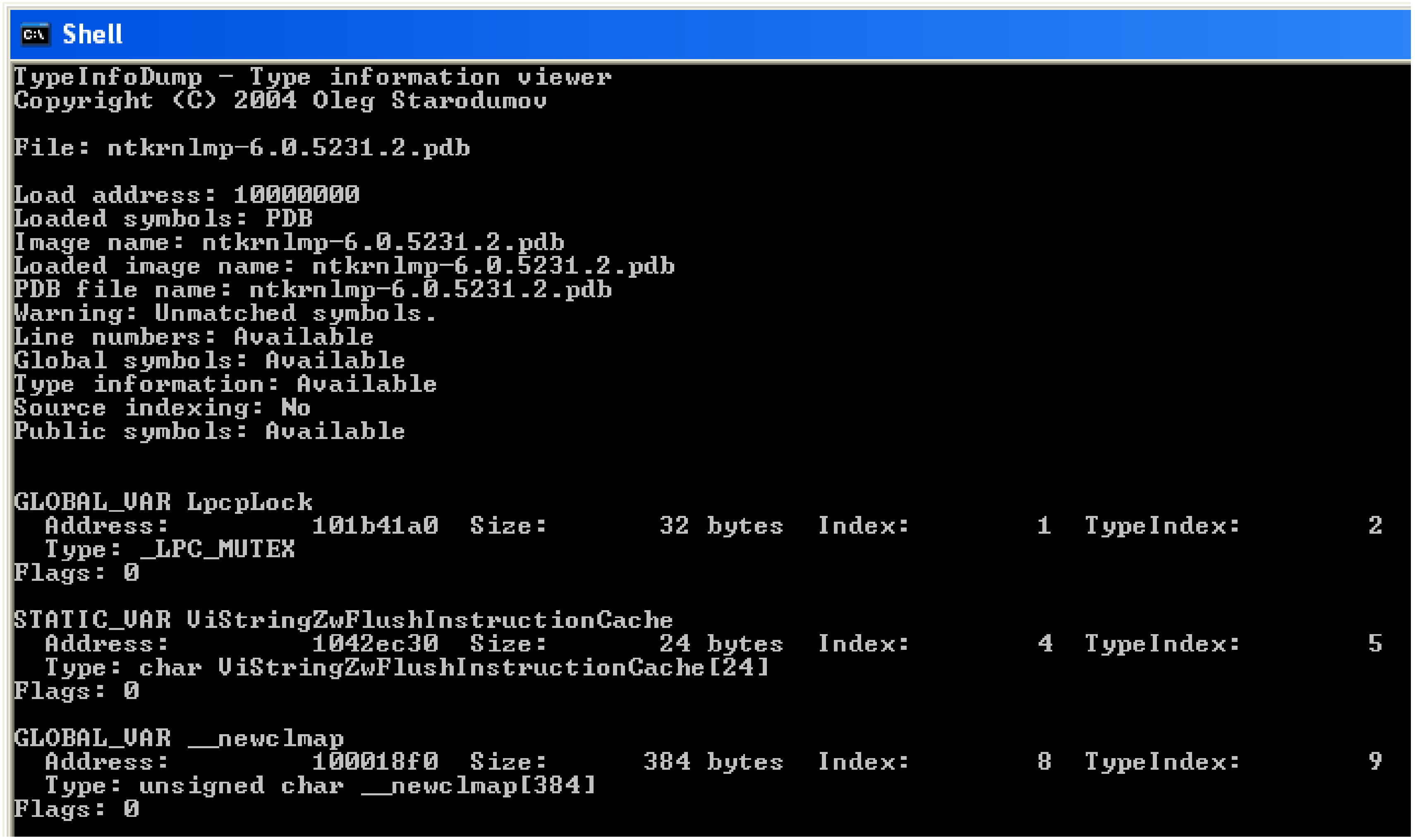
- IDA Pro Disassembler by Hex-Rays
  http://www.hex-rays.com/idapro/

Extending Profiles
Research Structure Information


Dump D:\Projekte\Windows Speicher\Sammlung\5.1.2600.3093.kernel.dmp - WinDbg:6.11.0001.404 X86
File Edit View Debug Window Help
Command

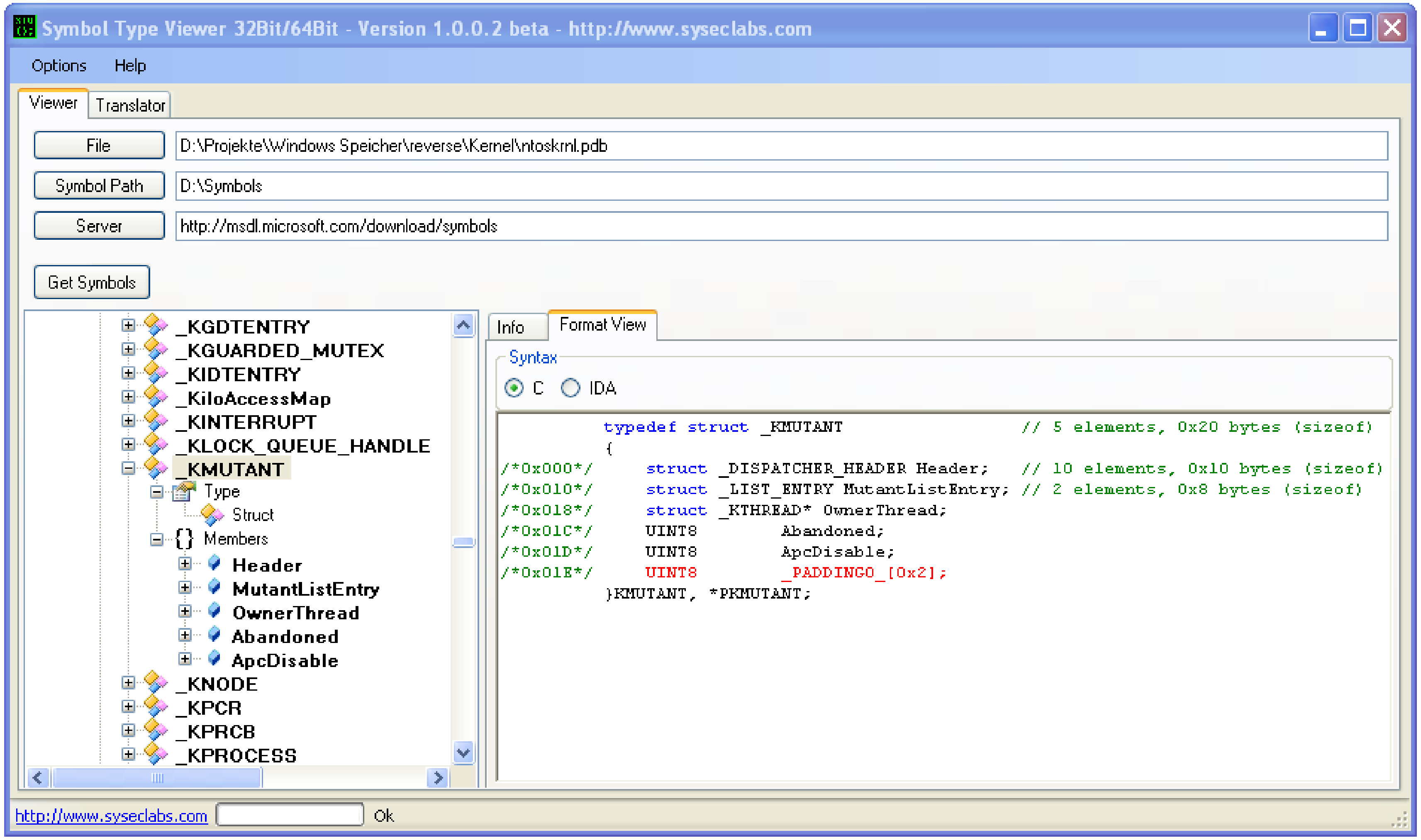
```
kd> dt -v nt!_EPROCESS
struct _EPROCESS, 107 elements, 0x260 bytes
   +0x000 Pcb              : struct _KPROCESS, 29 elements, 0x6c bytes
   +0x06c ProcessLock      : struct _EX_PUSH_LOCK, 5 elements, 0x4 bytes
   +0x070 CreateTime       : union _LARGE_INTEGER, 4 elements, 0x8 bytes
   +0x078 ExitTime         : union _LARGE_INTEGER, 4 elements, 0x8 bytes
   +0x080 RundownProtect   : struct _EX_RUNDOWN_REF, 2 elements, 0x4 bytes
   +0x084 UniqueProcessId  : Ptr32 to Void
   +0x088 ActiveProcessLinks : struct _LIST_ENTRY, 2 elements, 0x8 bytes
   +0x090 QuotaUsage       : [3] Uint4B
   +0x09c QuotaPeak        : [3] Uint4B
   +0x0a8 CommitCharge     : Uint4B
   +0x0ac PeakVirtualSize  : Uint4B
   +0x0b0 VirtualSize      : Uint4B
   +0x0b4 SessionProcessLinks : struct _LIST_ENTRY, 2 elements, 0x8 bytes
   +0x0bc DebugPort        : Ptr32 to Void
   +0x0c0 ExceptionPort    : Ptr32 to Void
   +0x0c4 ObjectTable      : Ptr32 to struct _HANDLE_TABLE, 14 elements, 0x44 bytes
```

Extending Profiles
Research Structure Information



```
TypeInfoDump - Type information viewer
Copyright (C) 2004 Oleg Starodumov

File: ntkrnlmp-6.0.5231.2.pdb

Load address: 10000000
Loaded symbols: PDB
Image name: ntkrnlmp-6.0.5231.2.pdb
Loaded image name: ntkrnlmp-6.0.5231.2.pdb
PDB file name: ntkrnlmp-6.0.5231.2.pdb
Warning: Unmatched symbols.
Line numbers: Available
Global symbols: Available
Type information: Available
Source indexing: No
Public symbols: Available


GLOBAL_VAR LpcpLock
  Address:        101b41a0  Size:       32 bytes  Index:        1  TypeIndex:        2
  Type: _LPC_MUTEX
Flags: 0

STATIC_VAR UiStringZwFlushInstructionCache
  Address:        1042ec30  Size:       24 bytes  Index:        4  TypeIndex:        5
  Type: char UiStringZwFlushInstructionCache[24]
Flags: 0

GLOBAL_VAR __newclmap
  Address:        100018f0  Size:      384 bytes  Index:        8  TypeIndex:        9
  Type: unsigned char __newclmap[384]
Flags: 0
```

Extending Profiles
Research Structure Information



File: D:\Projekte\Windows Speicher\reverse\Kernel\ntoskrnl.pdb

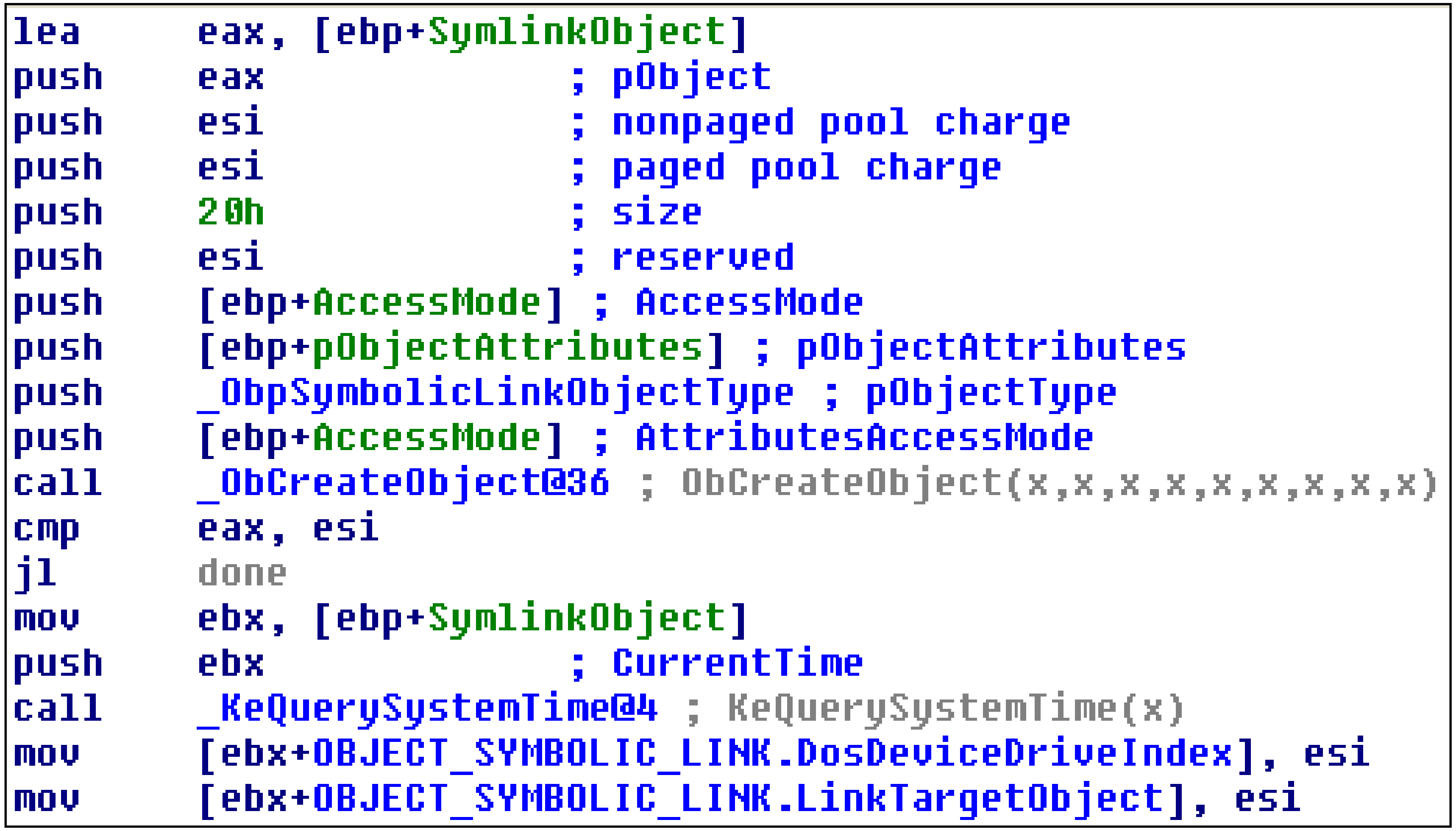
Symbol Path: D:\Symbols

Server: http://msdl.microsoft.com/download/symbols

Syntax: [✓ C] IDA

```
          typedef struct _KMUTANT                   // 5 elements, 0x20 bytes (sizeof)
          {
/*0x000*/     struct _DISPATCHER_HEADER Header;    // 10 elements, 0x10 bytes (sizeof)
/*0x010*/     struct _LIST_ENTRY MutantListEntry; // 2 elements, 0x8 bytes (sizeof)
/*0x018*/     struct _KTHREAD* OwnerThread;
/*0x01C*/     UINT8        Abandoned;
/*0x01D*/     UINT8        ApcDisable;
/*0x01E*/     UINT8        _PADDING0_[0x2];
          }KMUTANT, *PKMUTANT;
```

Extending Profiles
Research Structure Information



```
lea     eax, [ebp+SymlinkObject]
push    eax                     ; pObject
push    esi                     ; nonpaged pool charge
push    esi                     ; paged pool charge
push    20h                     ; size
push    esi                     ; reserved
push    [ebp+AccessMode] ; AccessMode
push    [ebp+pObjectAttributes] ; pObjectAttributes
push    _ObpSymbolicLinkObjectType ; pObjectType
push    [ebp+AccessMode] ; AttributesAccessMode
call    _ObCreateObject@36 ; ObCreateObject(x,x,x,x,x,x,x,x,x)
cmp     eax, esi
jl      done
mov     ebx, [ebp+SymlinkObject]
push    ebx                     ; CurrentTime
call    _KeQuerySystemTime@4 ; KeQuerySystemTime(x)
mov     [ebx+OBJECT_SYMBOLIC_LINK.DosDeviceDriveIndex], esi
mov     [ebx+OBJECT_SYMBOLIC_LINK.LinkTargetObject], esi
```

Extending Profiles
Define the structure

```
symlink_types = {
    '_SYMLINK_OBJECT' : [ 0x20, {
        'CreatedTime' : [ 0x0, ['_KSYSTEM_TIME']],
        'Target' : [ 0x8, ['_UNICODE_STRING']],
        'LinkTargetRemaining' : [ 0x10, ['_UNICODE_STRING']],
        'LinkTargetObject': [ 0x18, ['pointer', ['void']]],
        'DosDeviceDriveIndex' : [ 0x1c, ['unsigned long']],
    } ],
}

# …
# merge type information
types.update(symlink_types)
```

Extending Profiles
Define the structure

- native types: see also `builtin_types` in `forensics/object.py`
  - char
  - unsigned char
  - unsigned short
  - short
  - int
  - unsigned int
  - long
  - unsigned long
  - long long
  - unsigned long long
  - address

- pointer:
  - `['pointer', ['_HANDLE_TABLE']]]`
  - `['pointer', ['void']]]`

- array: `['array', 16,['unsigned char']]]`

Files and Functions

Files and Functions
Directories

./ (base directory)

- administrative stuff (readme, license, setup.py)
- main script (volatility)
- supporting core files (vmodules, vsyms, vtypes, vutils)

./forensics/

- x86 address translation
- Volatility registry
- base classes (address spaces, plugins)

Files and Functions
Directories

./forensics/win32/

- more address spaces (crash dump, hibernate file)
- constrained-based scanners
- fast pool scanner

./memory_objects/

- drop data structures and objects here, recursively searched

./memory_plugins/

- drop your plug-ins here, recursively searched

./thirdparty/

- utility functions taken from other projects

Building Blocks

Plug-ins

Writing Plugins
Create a new class

- Subclass from forensics.commands.command
- The name of the class becomes your new command verb
- There can be multiple classes (and commands) in a single plugin file.

```
class mycmd(forensics.commands.command):
```

Writing Plugins
Provide meta-information and help

```
    # Declare meta information associated with this plugin

    meta_info = forensics.commands.command.meta_info
    meta_info['author'] = 'Your Name'
    meta_info['copyright'] = 'Copyright (c) 2009 Your Name'
    meta_info['contact'] = 'your_name@example.com'
    meta_info['license'] = 'GNU General Public License 2.0 or later'
    meta_info['url'] = 'http://www.example.com//'
    meta_info['os'] = 'WIN_32_XP_SP2'
    meta_info['version'] = '1.0'

   def help(self):
        return  “list foobar objects"
```

Writing Plugins
Optional: add command line options

```
def parser(self):
    # call method in superclass
    forensics.commands.command.parser(self)

    # add your own options, first a string
    self.op.add_option(‘-o’, ‘–offset’, help=‘Offset (in hex)’,
        action=‘store’, type=‘string’, dest=‘offset’)

    # and now a boolean value
    self.op.add_option(‘-v’, ‘–verbose’, help=‘print more information’,
        action=‘store_true’, dest=‘verbosity’)
```

- Volatility command line parser builds on the `optparse` module.
- For further documentation and examples see the Python library docs at http://docs.python.org/library/optparse.html

Writing Plugins
Do all the work

```
def execute(self):
    op = self.op                # command line parser instance
    opts = self.opts            # parsed options

    # work hard
    # …

    # display results
    print "%20s %6s %6s“ % ('Name', 'Pid' , 'PPid')
```

Writing Plugins
A peek into the future

Meta info

- meta_info is likely to go away

Rendering

- separation of calculations and rendering steps
- single `calculate()` routine
- specialized renderers, named `render_`*`format`*`()`
- `execute()` calls `calculate`(), then the appropriate renderer
- standard option will select the format, defaults to “text”

Writing Plugins
Hands-on: Write your first plug-in

Create a plug-in named “myplugin.py” that writes “Hello world!” to the console.

Writing Plugins
Hands-on: Write your first plug-in

```
class mycmd(forensics.commands.command):
    meta_info = forensics.commands.command.meta_info
    meta_info['author'] = 'Your Name'
    meta_info['copyright'] = 'Copyright (c) 2009 Your Name'
    meta_info['contact'] = 'your_name@example.com'
    meta_info['license'] = 'GNU General Public License 2.0 or later'
    meta_info['url'] = 'http://www.example.com//'
    meta_info['os'] = 'WIN_32_XP_SP2'
    meta_info['version'] = '1.0‘

  def help(self):
    return  “Prints a famous greeting."

  def execute(self):
    print “Hello world!”
```

Writing Plugins
Hands-on: Write your first plug-in

- Modify your plug-in to
    - accept a numeric parameter “-a”,
    - store it in a variable “myaddr” and
    - echo it to the console.
- Test it!

Writing Plugins

Hands-on: Write your first plug-in

```
class mycmd(forensics.commands.command):
    meta_info = forensics.commands.command.meta_info
    meta_info['author'] = 'Your Name'
    meta_info['copyright'] = 'Copyright (c) 2009 Your Name'
    meta_info['contact'] = 'your_name@example.com'
    meta_info['license'] = 'GNU General Public License 2.0 or later'
    meta_info['url'] = 'http://www.example.com//'
    meta_info['os'] = 'WIN_32_XP_SP2'
    meta_info['version'] = '1.0‘

  def help(self):
    return  “Prints a famous greeting.”

  def parser(self):
    forensics.commands.command.parser(self)
    self.op.add_option(‘-a’, action=’store’, type=’int’, dest=’myaddr’)

  def execute(self):
    op = self.op        # command line parser instance
    opts = self.opts    # parsed options
    print “The value is %x” % self.opts.myaddr
```

Writing Plugins

Hands-on: Write your first plug-in

- Modify your plug-in to
  - load an image file (-f)
  - convert the virtual address (-a) into a physical address and
  - echo it to the console.

Writing Plugins

Hands-on: Write your first plug-in

```python
def help(self):
    return  “Convert virtual into physical address”

  def parser(self):
    forensics.commands.command.parser(self)
    self.op.add_option(‘-a’, action=’store’, type=’int’, dest=’myaddr’)

  def execute(self):
    op = self.op       # command line parser instance
    opts = self.opts   # parsed options

    (addr_space, , ) = load_and_identify_image(self.op, self.opts)
    print “%x -> %x” % (self.opts.myaddr,
                          addr_space.vtop(self.opts.myaddr))
```

Thank You for Your Attention!

Andreas Schuster

a.schuster@yendor.net
http://computer.forensikblog.de/en/