Windows Memory Forensics with Volatility

Andreas Schuster
About the Tutorial
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory fundamentals</td>
<td>Volatility overview</td>
<td>Address spaces</td>
</tr>
<tr>
<td>Memory acquisition techniques</td>
<td>Built-in functions</td>
<td>Objects and Profiles</td>
</tr>
<tr>
<td>Kernel objects</td>
<td>Selected plug-ins</td>
<td>Your first plug-in</td>
</tr>
<tr>
<td>Memory analysis techniques</td>
<td>Hands-on exercises</td>
<td>Building blocks</td>
</tr>
</tbody>
</table>
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.volatile systems.com/

- Brendan Dolan-Gavitt
  Georgia Institute of Technology, Atlanta, GA, USA
  http://moyix.blogspot.com/

- Bradley Schatz
  Schatz Forensic Pty Ltd, Brisbane, Australia

- hogfly
  http://forensicir.blogspot.com/
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!
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.
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

<table>
<thead>
<tr>
<th>Action</th>
<th>% RAM unchanged</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>256 MB RAM</td>
</tr>
<tr>
<td>Start</td>
<td>100.0</td>
</tr>
<tr>
<td>Idle for 1 hour</td>
<td>90.4</td>
</tr>
<tr>
<td>Idle for 2 hours</td>
<td>79.7</td>
</tr>
<tr>
<td>DD (live acquisition)</td>
<td>76.9</td>
</tr>
<tr>
<td>Idle for 15 hours</td>
<td>74.8</td>
</tr>
<tr>
<td>WFT (live response)</td>
<td>67.2</td>
</tr>
<tr>
<td></td>
<td>512 MB RAM</td>
</tr>
<tr>
<td>Start</td>
<td>100.0</td>
</tr>
<tr>
<td>Idle for 1 hour</td>
<td>96.7</td>
</tr>
<tr>
<td>Idle for 2 hours</td>
<td>96.1</td>
</tr>
<tr>
<td>DD (live acquisition)</td>
<td>89.8</td>
</tr>
<tr>
<td>Idle for 15 hours</td>
<td>85.6</td>
</tr>
<tr>
<td>WFT (live response)</td>
<td>69.4</td>
</tr>
</tbody>
</table>

Effects on main memory, according to Walters and Petroni (2006)
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)
Preserve Data in Order of Volatility

- CD
- Printout
- Backup
- Harddrive
- Processes
- Network Status
- Main Memory
- Register

Data Lifespan in Seconds ($\log_{10}$) according to Venema and Farmer (2004)
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)
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
“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
    ```
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

- Different levels of compression

- Meta-Information (case number, examiner, MD5 hash, etc.)

- Similar, but open source: Advanced Forensic Format (AFF)
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
  - Enables access to physical memory over iSCSI
  - Use with acquisition tool of your choice

- 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
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
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.
- 4 GiB of (virtual) address space per process

- Split into halves

![Diagram showing 4 GiB address space split into Application (2 GiB) and System (2 GiB) sections with explorer.exe process]
Physical memory is divided into so called “pages”.

Allocated virtual memory is mapped onto physical memory page by page.

```
sol.exe
explorer.exe
```

physical memory
The same page of physical memory can appear at different locations within the same address space or in different address spaces.
Data can be moved from physical memory into a page file to clear some space.
Memory Pools
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)
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.
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
- 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 resources
  - Driver: loadable kernel module
  - File: instance of an open file or I/O device
  - Token: SID and privileges
  - Key: registry
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
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!
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)
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
  - Invalid 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/
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](http://www.activestate.com/activepython)

- Volatility
  - stable [https://www.volatilesystems.com/default/volatility](https://www.volatilesystems.com/default/volatility)
  - SVN on [http://code.google.com/p/volatility/](http://code.google.com/p/volatility/), see instructions

- Plug-ins may require additional software, e.g.
  - pydasm [http://dkbza.org/pydasm.html](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
For a list of internal- and plug-in commands:

```
python volatility
```

For help on any command:

```
python volatility command --help
```
-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.
Hands-on: Information about the Memory Image

- Analyze memory image “/samples/exemplar13.vmem” by hogfly.
- Authenticate the memory image
  MD5 5ec0c6dffaa29b1bd5a6cbec1829df25d
- Determine the OS version and the system’s time. This becomes the endpoint of our timeline.
Authenticate the memory image
   MD5  5ec0c6dfffa29b1bd5a6cbec1829df25d

md5sum /samples/exemplar13.vmem
5ec0c6dfffa29b1bd5a6cbec1829df25d

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

```bash
> python volatility ident -f /samples/exemplar13.vmem
  Image Name: /samples/exemplar13.vmem
  Image Type: Service Pack 2
    VM Type: pae
      DTB: 0x7d0000

> 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
```
| 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
  - \(-s\) \textit{HEXADDRESS}
    - \(--\text{start}=\text{HEXADDRESS}\)
    Start address

- \(-e\) \textit{HEXADDRESS}
  - \(--\text{end}=\text{HEXADDRESS}\)
  End address

- \(-s\)
  - \(--\text{slow}\)
  Perform scan on original address space instead of flat file
### Output format

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

<table>
<thead>
<tr>
<th>No.</th>
<th>PID</th>
<th>TID</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>888</td>
<td>1716</td>
<td>0x0008a020</td>
</tr>
<tr>
<td>2</td>
<td>888</td>
<td>1712</td>
<td>0x0008ada8</td>
</tr>
<tr>
<td>3</td>
<td>1296</td>
<td>1384</td>
<td>0x001a5230</td>
</tr>
</tbody>
</table>

- Version on DVD also reports thread creation and exit times.
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
F POV 7

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
- **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
  - \(-s\) \textit{HEXADDRESS}
    - \(--\text{start}=\text{HEXADDRESS}\)
      Start address
  - \(-e\) \textit{HEXADDRESS}
    - \(--\text{end}=\text{HEXADDRESS}\)
      End address
  - \(-s\)
    - \(--\text{slow}\)
      Perform scan on original address space instead of flat file

- psscan and psscan2
  - \(-d\) \textit{FILE}
    - \(--\text{dot}=\text{FILE}\)
      Draw process tree in DOT format for GraphViz
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  

- 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)
<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>PPid</th>
<th>Thds</th>
<th>Hnds</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x81292780: System</td>
<td>4</td>
<td>-1</td>
<td>49</td>
<td>222</td>
<td>Thu Jan 01 00:00:00 1970</td>
</tr>
<tr>
<td>. 0x811A5978: smss.exe</td>
<td>432</td>
<td>4</td>
<td>3</td>
<td>21</td>
<td>Thu Jun 11 14:31:40 2009</td>
</tr>
<tr>
<td>.. 0x811175A8: winlogon.exe</td>
<td>512</td>
<td>432</td>
<td>18</td>
<td>515</td>
<td>Thu Jun 11 14:31:47 2009</td>
</tr>
<tr>
<td>... 0xFFBA0228: services.exe</td>
<td>556</td>
<td>512</td>
<td>15</td>
<td>259</td>
<td>Thu Jun 11 14:31:50 2009</td>
</tr>
<tr>
<td>.... 0x811C6A10: svchost.exe</td>
<td>1000</td>
<td>556</td>
<td>5</td>
<td>57</td>
<td>Thu Jun 11 14:32:02 2009</td>
</tr>
<tr>
<td>.... 0x8110C1A8: vmachlp.exe</td>
<td>744</td>
<td>556</td>
<td>1</td>
<td>24</td>
<td>Thu Jun 11 14:31:54 2009</td>
</tr>
<tr>
<td>.... 0xFFAA3B0: netdte.exe</td>
<td>1236</td>
<td>556</td>
<td>10</td>
<td>68</td>
<td>Thu Jun 11 14:32:07 2009</td>
</tr>
<tr>
<td>.... 0xFFB937E8: VMMservice.exe</td>
<td>1332</td>
<td>556</td>
<td>3</td>
<td>162</td>
<td>Thu Jun 11 14:32:10 2009</td>
</tr>
<tr>
<td>.... 0x8110F900: spoolsv.exe</td>
<td>1100</td>
<td>556</td>
<td>14</td>
<td>124</td>
<td>Thu Jun 11 14:32:03 2009</td>
</tr>
<tr>
<td>.... 0x810B17E8: svchost.exe</td>
<td>864</td>
<td>556</td>
<td>10</td>
<td>213</td>
<td>Thu Jun 11 14:32:00 2009</td>
</tr>
<tr>
<td>.... 0xFFEB9D30: svchost.exe</td>
<td>928</td>
<td>556</td>
<td>56</td>
<td>1334</td>
<td>Thu Jun 11 14:32:00 2009</td>
</tr>
<tr>
<td>.... 0xFF96DA0: alg.exe</td>
<td>1524</td>
<td>556</td>
<td>6</td>
<td>103</td>
<td>Thu Jun 11 14:32:14 2009</td>
</tr>
<tr>
<td>.... 0xFFBA47E8: svchost.exe</td>
<td>792</td>
<td>556</td>
<td>18</td>
<td>164</td>
<td>Thu Jun 11 14:31:59 2009</td>
</tr>
<tr>
<td>.... 0xFFBCFA20: svchost.exe</td>
<td>1036</td>
<td>556</td>
<td>7</td>
<td>122</td>
<td>Thu Jun 11 14:32:02 2009</td>
</tr>
<tr>
<td>... 0xFFBA9558: lsass.exe</td>
<td>568</td>
<td>512</td>
<td>15</td>
<td>295</td>
<td>Thu Jun 11 14:31:51 2009</td>
</tr>
<tr>
<td>.. 0x810B1C08: csrss.exe</td>
<td>488</td>
<td>432</td>
<td>12</td>
<td>329</td>
<td>Thu Jun 11 14:31:45 2009</td>
</tr>
<tr>
<td>Name</td>
<td>Pid</td>
<td>PPid</td>
<td>Thds</td>
<td>Hnds</td>
<td>Time</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>0x81292780:System</td>
<td>4</td>
<td>-1</td>
<td>49</td>
<td>222</td>
<td>Thu Jan 01 00:00:00 1970</td>
</tr>
<tr>
<td>. 0x8115978:smss.exe</td>
<td>432</td>
<td>4</td>
<td>3</td>
<td>21</td>
<td>Thu Jun 11 14:31:40 2009</td>
</tr>
<tr>
<td>path: \SystemRoot\System32\smss.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit: \Device\HarddiskVolume1\WINDOWS\system32\smss.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.. 0x811175A8:winlogon.exe</td>
<td>512</td>
<td>432</td>
<td>18</td>
<td>515</td>
<td>Thu Jun 11 14:31:47 2009</td>
</tr>
<tr>
<td>cmd: None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path: None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit: \Device\HarddiskVolume1\WINDOWS\system32\winlogon.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>... 0xFFBA0228:services.exe</td>
<td>556</td>
<td>512</td>
<td>15</td>
<td>259</td>
<td>Thu Jun 11 14:31:50 2009</td>
</tr>
<tr>
<td>cmd: C:\WINDOWS\system32\services.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path: C:\WINDOWS\system32\services.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit: \Device\HarddiskVolume1\WINDOWS\system32\services.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.... 0x811C6A10:svchost.exe</td>
<td>1000</td>
<td>556</td>
<td>5</td>
<td>57</td>
<td>Thu Jun 11 14:32:02 2009</td>
</tr>
<tr>
<td>cmd: C:\WINDOWS\system32\svchost.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path: C:\WINDOWS\system32\svchost.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit: \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.... 0x811OCA8:vmacthlp.exe</td>
<td>744</td>
<td>556</td>
<td>1</td>
<td>24</td>
<td>Thu Jun 11 14:31:54 2009</td>
</tr>
<tr>
<td>cmd: &quot;C:\Program Files\VMware\VMware Tools\vmacthlp.exe&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path: C:\Program Files\VMware\VMware Tools\vmacthlp.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit: \Device\HarddiskVolume1\Program Files\VMware\VMware Tools\vmacthlp.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.... 0xFFAA3B0:netdde.exe</td>
<td>1236</td>
<td>556</td>
<td>10</td>
<td>68</td>
<td>Thu Jun 11 14:32:07 2009</td>
</tr>
<tr>
<td>cmd: C:\WINDOWS\system32\netdde.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path: C:\WINDOWS\system32\netdde.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audit: \Device\HarddiskVolume1\WINDOWS\system32\netdde.exe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- Analyze memory image “/samples/exemplar13.vmem” by hogfly.

- Find the PID, start/end times and exit code for processes
  - explorer.exe
  - ud32.exe
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu Jan 08 01:53:09 2009</td>
<td>processes 464 and 1040 (ud32.exe) started by process 1928 (explorer.exe)</td>
</tr>
<tr>
<td>Thu Jan 08 01:53:10 2009</td>
<td>process 1040 terminated, exit code 0</td>
</tr>
<tr>
<td>Thu Jan 08 01:54:57 2009</td>
<td>memory image obtained</td>
</tr>
</tbody>
</table>
dlllist

-Enumerates DLLs (and EXEs) loaded by a process
-Does not work for terminated or hidden processes

-\texttt{-p PID} \\
-\texttt{--pid=PID}

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

\begin{tabular}{lll}
Base & Size & Path \\
0x1000000 & 0xff000 & C:\WINDOWS\Explorer.EXE \\
0x7c900000 & 0xb0000 & C:\WINDOWS\system32\ntdll.dll \\
0x7c800000 & 0xf4000 & C:\WINDOWS\system32\kernel32.dll \\
\end{tabular}
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
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.exe (1332): S-1-5-18 (Local System)
VMwareService.exe (1332): S-1-5-32-544 (Administrators)
VMwareService.exe (1332): S-1-1-0 (Everyone)
VMwareService.exe (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
**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
Output format
- Unique Process ID (PID)
- Port (if applicable)
- Protocol
- Create time

Output formats differ slightly.
## sockets

<table>
<thead>
<tr>
<th>Pid</th>
<th>Port</th>
<th>Proto</th>
<th>Create Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1026</td>
<td>6</td>
<td>Thu Jun 11 14:32:15 2009</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>47</td>
<td>Thu Jun 11 14:32:15 2009</td>
</tr>
<tr>
<td>928</td>
<td>0</td>
<td>2</td>
<td>Thu Jun 11 14:32:13 2009</td>
</tr>
<tr>
<td>4</td>
<td>445</td>
<td>6</td>
<td>Thu Jun 11 14:31:28 2009</td>
</tr>
</tbody>
</table>

## sockscan / sockscan2

<table>
<thead>
<tr>
<th>PID</th>
<th>Port</th>
<th>Proto</th>
<th>Create Time</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1524</td>
<td>1025</td>
<td>6</td>
<td>Thu Jun 11 14:32:15 2009</td>
<td>0x0083c838</td>
</tr>
<tr>
<td>4</td>
<td>1026</td>
<td>6</td>
<td>Thu Jun 11 14:32:15 2009</td>
<td>0x01031620</td>
</tr>
<tr>
<td>1640</td>
<td>31337</td>
<td>6</td>
<td>Thu Jun 11 14:35:15 2009</td>
<td>0x0104eb78</td>
</tr>
<tr>
<td>4</td>
<td>138</td>
<td>17</td>
<td>Thu Jun 11 14:32:06 2009</td>
<td>0x01057e98</td>
</tr>
</tbody>
</table>
Connections
- Locates tcpipl module
- Looks for TCBtable at known offsets into module
- Locates and dumps connection objects

connscan / connscan2
- Searches for POOL_HEADER
- connscan2 is much faster!
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
Output format
- Local IP address and port
- Remote IP address and port
- Unique Process ID (PID)

Output formats differ slightly.
### connections

<table>
<thead>
<tr>
<th>Local Address</th>
<th>Remote Address</th>
<th>Pid</th>
</tr>
</thead>
</table>

### connscan / connscan2

<table>
<thead>
<tr>
<th>Local Address</th>
<th>Remote Address</th>
<th>Pid</th>
</tr>
</thead>
</table>
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)
<table>
<thead>
<tr>
<th>Time</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu Jan 08 01:53:07 2009</td>
<td>process 1928 (explorer.exe) creates socket for port 1048/tcp, connects to 67.215.11.138:7000</td>
</tr>
<tr>
<td>Thu Jan 08 01:53:09 2009</td>
<td>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)</td>
</tr>
<tr>
<td>Thu Jan 08 01:53:10 2009</td>
<td>process 464 creates sockets for ports 27714/tcp and 1052/udp process 1040 terminated, exit code 0</td>
</tr>
<tr>
<td>Thu Jan 08 01:54:57 2009</td>
<td>memory image obtained</td>
</tr>
</tbody>
</table>

**Handson:** Timeline
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\PARAMETERS\S\PROTOCOL_CATALOG9
VolReg plug-in package

- Written by Brendan Dolan-Gavitt

- 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!
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.
python volatility hivescan -f /samples/exemplar13.vmem

Offset (hex)
34786144 0x212cb60
35029896 0x2168388
52190048 0x31c5b60
61227776 0x3a64300
62263304 0x3b61008
62692192 0x3bc9b60
78032904 0x4a6b008
117499936 0x700e820
117721952 0x7044b60
118016032 0x708c820
181174280 0xacc8008
182220832 0xadc7820
### Commands

**Hands-on: Registry**

```plaintext
python volatility hivelist -f /samples/exemplar13.vmem -o 0x212cb60
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xe179e008</td>
<td>[no name]</td>
</tr>
<tr>
<td>0xe1a58b60</td>
<td>\Documents and Settings\foo\NTUSER.DAT</td>
</tr>
<tr>
<td>0xe1548008</td>
<td>[no name]</td>
</tr>
<tr>
<td>0xe1535820</td>
<td>\Documents and Settings\LocalService\NTUSER.DAT</td>
</tr>
<tr>
<td>0xe1095820</td>
<td>[no name]</td>
</tr>
<tr>
<td>0xe107e820</td>
<td>\Documents and Settings\NetworkService\NTUSER.DAT</td>
</tr>
<tr>
<td>0xe13a3008</td>
<td>\WINDOWS\system32\config\software</td>
</tr>
<tr>
<td>0xe1397300</td>
<td>\WINDOWS\system32\config\default</td>
</tr>
<tr>
<td>0xe13a0b60</td>
<td>\WINDOWS\system32\config\SECURITY</td>
</tr>
<tr>
<td>0xe1362b60</td>
<td>\WINDOWS\system32\config\SAM</td>
</tr>
<tr>
<td>0xe11c2008</td>
<td>[no name]</td>
</tr>
<tr>
<td>0xe1018388</td>
<td>\WINDOWS\system32\config\system</td>
</tr>
<tr>
<td>0xe1008b60</td>
<td>[no name]</td>
</tr>
</tbody>
</table>
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)
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)
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)
Create a timeline of events for the whole registry.

```bash
> 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
```
MANDIANT Highlighter
http://www.mandiant.com/software/highlighter.htm
<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thu Jan 08 01:52:50 2009</td>
<td><a href="http://192.168.30.129/malware/sys32.exe">http://192.168.30.129/malware/sys32.exe</a> executed sys32.exe and flypaper.exe saved to foo’s desktop</td>
</tr>
<tr>
<td>Thu Jan 08 01:53:07 2009</td>
<td>process 1928 (explorer.exe) creates socket for port 1048/tcp, connects to 67.215.11.138:7000 sys32.exe entry for Active Setup</td>
</tr>
<tr>
<td>Thu Jan 08 01:53:09 2009</td>
<td>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)</td>
</tr>
<tr>
<td>Thu Jan 08 01:53:10 2009</td>
<td>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</td>
</tr>
<tr>
<td>Thu Jan 08 01:54:57 2009</td>
<td>memory image obtained</td>
</tr>
</tbody>
</table>
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
- cryptoscan
  - by Jesse Kornblum
  - finds TrueCrypt passphrases

- suspicious
  - by Jesse Kernblum
  - searches for suspicious command line parameters
keyboardbuffer
- by Andreas Schuster
  [Link](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!
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
- Analyze the memory image “exemplar13.vmem” by hogfly.

- Dump the LM/NTLM hashes and examine their quality
Commands
Hands-on: Secrets
- **malfind**
  - by Michael Hale Ligh
  - Looks for (possibly) injected code
  - Invoke from Volatility base directory only!

- **usermode_hooks**
  - by Michael Hale Ligh
  - Detects IAT and EAT hooks, detours
  - Depends on pydasm and pefile
ssdt

- 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
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
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
BaseAddressSpace
  FileAddressSpace
    BufferAddressSpace
    EWFAddressSpace
    WindowsCrashDumpSpace32
    WindowsHiberFileSpace32
  IA32PagedMemory
    IA32PagedMemoryPae
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)`
How do you access data

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

<table>
<thead>
<tr>
<th>IA32PagedMemory</th>
<th>provides virtual address space, no PAE, CR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindowsHiberFileSpace32</td>
<td>decompresses file, provides physical address space</td>
</tr>
<tr>
<td>FileAddressSpace</td>
<td>hiberfil.sys</td>
</tr>
</tbody>
</table>
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
Dump debug symbols (PDB)

- Microsoft Debugger

- Symbol Type Viewer by Lionel d'Hauenens
  [http://www.labo-asso.com/download/SymbolTypeViewer_v1.0_beta.zip](http://www.labo-asso.com/download/SymbolTypeViewer_v1.0_beta.zip)

- TypeInfoDump by Oleg Starodumov:

Reverse-engineer kernel and drivers

- IDA Pro Disassembler by Hex-Rays
### Extending Profiles

Research Structure Information

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pcb</td>
<td>0x000</td>
<td>First element of process structure</td>
</tr>
<tr>
<td>ProcessLock</td>
<td>0x008</td>
<td>Represents the process lock</td>
</tr>
<tr>
<td>CreateTime</td>
<td>0x010</td>
<td>Holds the create time of the process</td>
</tr>
<tr>
<td>ExitTime</td>
<td>0x018</td>
<td>Stores the exit time of the process</td>
</tr>
<tr>
<td>RunDownProtect</td>
<td>0x020</td>
<td>Indicates if the process has run down</td>
</tr>
<tr>
<td>UniqueProcessId</td>
<td>0x028</td>
<td>Unique identifier for the process</td>
</tr>
<tr>
<td>ActiveProcessLinks</td>
<td>0x030</td>
<td>Links to active processes</td>
</tr>
<tr>
<td>QuoteUsage</td>
<td>0x038</td>
<td>Represents the process's quote usage</td>
</tr>
<tr>
<td>CommitCharge</td>
<td>0x040</td>
<td>Stores the commit charge for the process</td>
</tr>
<tr>
<td>PeakVirtualSize</td>
<td>0x048</td>
<td>Represents the peak virtual size of the process</td>
</tr>
<tr>
<td>VirtualSize</td>
<td>0x050</td>
<td>Stores the virtual size of the process</td>
</tr>
<tr>
<td>SessionProcessLinks</td>
<td>0x058</td>
<td>Links to sessions linked to the process</td>
</tr>
<tr>
<td>DebugPort</td>
<td>0x060</td>
<td>Pointer to the debug port of the process</td>
</tr>
<tr>
<td>ExceptionPort</td>
<td>0x068</td>
<td>Pointer to the exception port of the process</td>
</tr>
<tr>
<td>ObjectTable</td>
<td>0x070</td>
<td>Pointer to the object table of the process</td>
</tr>
</tbody>
</table>
Extending Profiles
Research Structure Information

```
Shell

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 ViStringZwFlushInstructionCache
  Address: 1042ec30  Size:  24 bytes  Index:  4  TypeIndex:  5
  Type: char ViStringZwFlushInstructionCache[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
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
j1     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

1. symlink_types = {
2.     '__SYMLINK_OBJECT' : [ 0x20, {
3.         'CreatedTime' : [ 0x0, ['_KSYSTEM_TIME']],
4.         'Target' : [ 0x8, ['_UNICODE_STRING']],
5.         'LinkTargetRemaining' : [ 0x10, ['_UNICODE_STRING']],
6.         'LinkTargetObject': [ 0x18, ['pointer', ['void']]],
7.         'DosDeviceDriveIndex' : [ 0x1c, ['unsigned long']],
8.     } ],
9. }
10.
11.# ...
12.# merge type information
13.types.update(symlink_types)
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
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.

1. class mycmd(forensics.commands.command):
# Declare meta information associated with this plugin

```python
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'
```

```python
def help(self):
    return "list foobar objects"
```
Writing Plugins
Optional: add command line options

```python
1. def parser(self):
2.     # call method in superclass
3.     forensics.commands.command.parser(self)
4.
5.     # add your own options, first a string
6.     self.op.add_option('-o', '--offset', help='Offset (in hex)',
7.     action='store', type='string', dest='offset')
8.
9.     # and now a boolean value
10.    self.op.add_option('-v', '--verbose', help='print more information',
11.       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](http://docs.python.org/library/optparse.html)
1. def execute(self):
2.     op = self.op  # command line parser instance
3.     opts = self.opts  # parsed options
4. 
5.     # work hard
6.     # ...
7. 
8.     # display results
9.     print "%20s %6s %6s" % ('Name', 'Pid', 'PPid')
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”
Create a plug-in named “myplugin.py” that writes “Hello world!” to the console.
1. `class mycmd(forensics.commands.command):
2.     meta_info = forensics.commands.command.meta_info
3.     meta_info['author'] = 'Your Name'
4.     meta_info['copyright'] = 'Copyright (c) 2009 Your Name'
5.     meta_info['contact'] = 'your_name@example.com'
6.     meta_info['license'] = 'GNU General Public License 2.0 or later'
7.     meta_info['url'] = 'http://www.example.com/'
8.     meta_info['os'] = 'WIN_32_XP_SP2'
9.     meta_info['version'] = '1.0'
10. 
11.    def help(self):
12.       return "Prints a famous greeting."
13. 
14.    def execute(self):
15.       print "Hello world!"
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

```python
1. class mycmd(forensics.commands.command):
2.     meta_info = forensics.commands.command.meta_info
3.     meta_info['author'] = 'Your Name'
4.     meta_info['copyright'] = 'Copyright (c) 2009 Your Name'
5.     meta_info['contact'] = 'your_name@example.com'
6.     meta_info['license'] = 'GNU General Public License 2.0 or later'
7.     meta_info['url'] = 'http://www.example.com//'
8.     meta_info['os'] = 'WIN_32_XP_SP2'
9.     meta_info['version'] = '1.0'
10.
11.    def help(self):
12.        return "Prints a famous greeting."
13.
14.    def parser(self):
15.        forensics.commands.command.parser(self)
16.        self.op.add_option('-a', action='store', type='int', dest='myaddr')
17.
18.    def execute(self):
19.        op = self.op # command line parser instance
20.        opts = self.opts # parsed options
21.        print "The value is %x" % self.opts.myaddr
```
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.
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/