Minding the Gap(s) in Memory Forensics

Golden G. Richard III

Professor of Computer Science
Associate Director for Cybersecurity at the Center for Computation and Technology (CCT)

golden@cct.lsu.edu / @nolaforensix
Academic

NSA Employee with TS/SCI Security Clearance

Owner of Private Security Firm

Lifelong "Hacker"

Applied Cybersecurity Education Advocate
Broad Areas of Interest in Cybersecurity

- Digital Forensics
- Memory Forensics
- Malware Analysis
- Reverse Engineering
- Exploit Development
- Penetration Testing
- Crypto
- Secure Software Engineering
- Social Engineering
- Network Security
- Usable Security
- SCADA

Copyright 2014-2017 by Golden G. Richard III (@nolaforensix)
Collaborators and Support

- Andrew Case
- Joe Sylve
- Lodovico Marziale
- Aisha Ali-Gombe
- Brendan Saltaformaggio
- …and others
- Some of the research in this talk supported by NSF
  - SaTC: CORE: Medium: Robust Memory Forensics Techniques for Userland Malware Analysis, Award # 1703683, PI: Golden G. Richard III, $1,113,426
Digital Forensics

“Tools and techniques to recover, preserve, and examine digital evidence stored on or transmitted by digital devices.”

Computers, PDAs, cellular phones, videogame consoles, digital cameras, copy machines, printers, digital voice recorders...
Where’s the Evidence?

Non-volatile Evidence

- Files and Deleted Files
- Filesystem metadata
- Application metadata
- Windows registry
- Print spool files
- Hibernation files
- Temp files
- Log files
- Slack space
- Swap files
- Browser caches
- Network traces

RAM: OS and app data structures

Memory forensics

Volatile Evidence
Some Data is Only in RAM

OS Kernel data structures
  Volatile registry branches
  Network connections
  Running processes
  Open files
  Clipboard data
  ...

Application data structures
  Encryption keys
  Plaintext for encrypted data
  Private browsing data
  ...

Application / malware code
Memory Analysis

Capture RAM from live system

• physical memory dumping tool
• VM memory snapshot
• VM introspection

Analyze Memory Dump

• strings
• carving
• Volatility / Rekall

Expose OS and Application Structures

• to yield useful evidence
Physical Memory Dumps

• A complete copy of RAM
• A collection of pages (4K on Intel)
• Must impose order using info in page tables—virtual \(\leftrightarrow\) physical address mapping—otherwise, essentially a random set of 4K chunks
Memory Analysis: 2004

$ grep -i murder /dev/mem

I loved Sally, but I murdered her in the park on...

Murder

murder

Murderous

You murdered my hamster!

Murdered
Memory Analysis: Then

• **strings / grep / etc.**
  – essentially no tools circa 2004

• **pt_finder (~2006)**
  – Windows process enumeration

• **FACE (~2008)**
  – Memory analysis framework created at UNO

• ...
Andreas Schuster’s Approach: pt_finder

• Simple process enumeration for Windows memory dumps
• Scan memory to find kernel structures describing processes and threads
  – Normal
  – Hidden
  – Terminated
• Don’t rely on kernel lists/tables
• Instead, search memory dump for objects that look like processes/threads
Schuster (2)

• Important ideas:
  – Windows pool allocator allocates memory for kernel objects
  – Pool allocator marks allocations with a 4-byte tag
  – Windows wraps additional metadata around important kernel structures
  – Kernel objects have an OBJECT_HEADER structure
  – Further, processes (EPROCESS) and threads (ETHREAD) have a DISPATCH_HEADER, used for scheduling/synchronization
  – Structure formats can be used to develop patterns

• Use these ideas to develop patterns for discovering interesting structures in a Windows memory dump

• Walk memory dump in 4K steps, searching each 4K block for matches
Pool allocation (ExAllocatePoolWithTag)

- **_POOL_HEADER**
- **_OBJECT_HEADER_PROCESS_INFO**
- **_OBJECT_HEADER_QUOTA_INFO**
- **_OBJECT_HEADER_HANDLE_INFO**
- **_OBJECT_HEADER_NAME_INFO**
- **_OBJECT_HEADER_CREATOR_INFO**

Optional headers

- Reference Counts, Security Descriptor, Object Type, Flags

- Size of allocation

- 0x30 Bytes

- Object size varies per structure

Object Body (e.g. _FILE_OBJECT)
Schuster: POOL_TAG

Fig. 1 – Definitions of the POOL_HEADER structure in Windows 2000 (left) and later versions (right).

PoolTag == 0xE36F7250 for processes (roughly, "Proc")
PoolTag == 0xE5726854 for threads (roughly, "Thre")
0x54687265 = Thre
Schuster: OBJECT_HEADER

Known values for live/dead processes and threads!

kd> dt_OBJECT_HEADER
+0x000  PointerCount : Int4B
+0x004  HandleCount : Int4B
+0x004  SEntry : Ptr32
+0x008  Type : Ptr32
+0x00c  NameInfoOffset : UChar
+0x00d  HandleInfoOffset : UChar
+0x00e  QuotaInfoOffset : UChar
+0x00f  Flags : UChar
+0x010  ObjectCreateInfo : Ptr32
+0x010  QuotaBlockCharged : Ptr32
+0x014  SecurityDescriptor : Ptr32

Fig. 2 – The OBJECT_HEADER structure provides information about an object’s instance.

Also know information about EPROCESS/ETHREAD structures, etc., which allow building a reasonable pattern.
Schuster’s PTfinder
Schuster’s PTfinder

```
552
MSTask.exe
started
2005-06-05
00:32:51
running

600
metasploit.exe
started
2005-06-05
00:55:08

600
metasploit.exe
exited
2005-06-05
00:55:08
code 0

788
metasploit.exe
started
2005-06-05
00:38:37

788
metasploit.exe
exited
2005-06-05
00:38:37
code 0
```
Volatility Framework

• Now, most work done with a physical memory dump tool + Volatility
• Most popular memory analysis framework
• Completely open source
• Portable, written in Python 😞
• Supports analysis of Windows, Linux, Mac, Android memory dumps
• Plugins add new functionality, build on vast amount of research that’s already been done
• Lots of existing plugins, fairly straightforward to add new ones
Memory Analysis: Now

Use plugins to analyze:

- Running processes
- Hidden processes
- Hooks that hide malware
- Network connections
- Encryption keys
- Private browsing data
- Clipboard data
- Volatile registry branches
- Command history
- Window hierarchy
+ "easily" develop plugins
Windows Process Lists

```bash
$ python vol.py -f lab.mem --profile=WinXPSP3x86 pslist

Volatility Foundation Volatility Framework 2.4
Offset (V) Name                  PID  PPID  Thds  Hnds  Sess  Start

0x823c8830  System              4    0    56    537   ---- 2013-03-14 03:02:22
0x81e7e180  smss.exe            580   4    3    19    ---- 2013-03-14 03:02:25
0x82315da0  csrss.exe           644   580   10    449    0  2013-03-14 03:02:26
0x81f37948  winlogon.exe        668   580   18    515    0  2013-03-14 03:02:27
0x81fec128  services.exe        712   668   15    281    0  2013-03-14 03:02:27

[snip]
0x81eb4300  vmtoolsd.exe        1684  1300   6    213    0  2013-03-14 03:02:45
0x8210b9c8  IEXPLORE.EXE         1764  1300   16    642    0  2013-03-14 03:03:04
0x81e79020  firefox.exe          180  1300   27    447    0  2013-03-14 03:03:05
0x81cb63d0  wuauclt.exe         1576  1072   3    104    0  2013-03-14 03:03:40
0x81e86bf8  alg.exe              1836  712    5    102    0  2013-03-14 03:04:00
0x8209eda0  wsCNTFY.exe          2672  1072   1    28    0  2013-03-14 03:04:01
0x82013340  jucheck.exe          2388  1656   2    104    0  2013-03-14 03:07:45
0x81e79418  thunderbird.exe     3832  1300   30    339    0  2013-03-14 03:12:54
0x8202b398  AcroRd32.exe         3684  180    0    ----    0  2013-03-14 14:19:16
0x81ecd3c0  cmd.exe              3812  3684   1    33    0  2013-03-14 14:19:29
0x81f55bd0  a[1].php             2280  3812   1    139    0  2013-03-14 14:19:30
0x8223b738  IEXPLORE.EXE         2276  2280   7    280    0  2013-03-14 14:19:32
0x822c8a58  AcroRd32.exe         2644  180    0    ----    0  2013-03-14 14:40:16
```
$ python vol.py -f lab.mem --profile=WinXPSP3x86 pstree
Volatility Foundation Volatility Framework 2.4

[snip]

<table>
<thead>
<tr>
<th>Process Name</th>
<th>PID</th>
<th>PPID</th>
<th>Type</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>explorer.exe</td>
<td>1300</td>
<td>1188</td>
<td>11</td>
<td>363</td>
</tr>
<tr>
<td>TSVNCache.exe</td>
<td>1556</td>
<td>1300</td>
<td>7</td>
<td>53</td>
</tr>
<tr>
<td>firefox.exe</td>
<td>180</td>
<td>1300</td>
<td>27</td>
<td>447</td>
</tr>
<tr>
<td>AcroRd32.exe</td>
<td>3684</td>
<td>180</td>
<td>0</td>
<td>-----</td>
</tr>
<tr>
<td>cmd.exe</td>
<td>3812</td>
<td>3684</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>a[1].php</td>
<td>2280</td>
<td>3812</td>
<td>1</td>
<td>139</td>
</tr>
<tr>
<td>IEXPLORE.EXE</td>
<td>2276</td>
<td>2280</td>
<td>7</td>
<td>280</td>
</tr>
</tbody>
</table>

Copyright 2014-2017 by Golden G. Richard III (@nolaforensix)
Cross-referencing OS Kernel Structures

• Malware can easily hide from analysis tools that rely on one information source
  • e.g., the kernel's process list

• Expand scope by cross-referencing kernel structures that provide information about executing processes
DKOM: FU “Rootkit” under Windows

Doubly-linked process list in Windows kernel

Example of DKOM: Direct Kernel Object Manipulation

C:\> fu –ph 2260

Processes continue to run because Windows scheduler handles threads, not processes
FU on PID 2260

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>PID</th>
<th>pslist</th>
<th>pscon</th>
<th>thrdproc</th>
<th>pspcid</th>
<th>csrss</th>
<th>session</th>
<th>deskthrd</th>
<th>ExitTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0501880</td>
<td>2260</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>0x0504880</td>
<td>2260</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x0501880</td>
<td>2260</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>0x0504880</td>
<td>2260</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Copyright 2014-2017 by Golden G. Richard III (@nolaforensix)
## Clipboard Contents

```bash
$ python vol.py --profile=Win7SP1x64 -f win7.vmem clipboard
```

Volatility Foundation Volatility Framework 2.4

<table>
<thead>
<tr>
<th>Session</th>
<th>WindowStation</th>
<th>Format</th>
<th>...</th>
<th>...</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CF_UNICODETEXT</td>
<td></td>
<td></td>
<td>I didn't kill Josephine!!!</td>
</tr>
</tbody>
</table>

...
Windows Command History

C:\>dir
Volume in drive C has no label.
Volume Serial Number is ACF0-0C63

Directory of C:\
06/10/2009 04:42 PM  24 autoexec.bat
06/10/2009 04:42 PM  10 config.sys
07/13/2009 09:37 PM  <DIR>  PerfLogs
08/25/2010 03:19 PM  <DIR>  Perl
02/17/2014 10:33 PM  <DIR>  Program Files

[snip]
consoles
### consoles

Screen 0x83c0 X:80 Y:4000
Dump:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Golden>pslist

pslist v1.29 - Sysinternals PList
Copyright (C) 2000-2009 Mark Russinovich
Sysinternals

Process information for ETOUFFE-VM:

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>Pri</th>
<th>Thd</th>
<th>Hnd</th>
<th>Priv</th>
<th>CPU Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0:19:35:171</td>
<td>0:00:00:000</td>
</tr>
<tr>
<td>System</td>
<td>4</td>
<td>8</td>
<td>102</td>
<td>575</td>
<td>132</td>
<td>0:00:18:735</td>
<td>1:58:24:698</td>
</tr>
<tr>
<td>smss</td>
<td>216</td>
<td>11</td>
<td>2</td>
<td>29</td>
<td>380</td>
<td>0:00:00:078</td>
<td>1:58:24:698</td>
</tr>
<tr>
<td>csrss</td>
<td>296</td>
<td>13</td>
<td>9</td>
<td>912</td>
<td>1964</td>
<td>0:00:01:014</td>
<td>1:58:21:890</td>
</tr>
<tr>
<td>csrss</td>
<td>336</td>
<td>13</td>
<td>12</td>
<td>506</td>
<td>16664</td>
<td>0:00:00:904</td>
<td>1:58:21:656</td>
</tr>
<tr>
<td>wininit</td>
<td>344</td>
<td>13</td>
<td>3</td>
<td>74</td>
<td>1308</td>
<td>0:00:00:280</td>
<td>1:58:21:625</td>
</tr>
<tr>
<td>winlogon</td>
<td>376</td>
<td>13</td>
<td>5</td>
<td>129</td>
<td>2844</td>
<td>0:00:00:436</td>
<td>1:58:21:547</td>
</tr>
<tr>
<td>services</td>
<td>452</td>
<td>9</td>
<td>17</td>
<td>293</td>
<td>5868</td>
<td>0:00:01:341</td>
<td>1:58:21:094</td>
</tr>
<tr>
<td>lsass</td>
<td>460</td>
<td>9</td>
<td>11</td>
<td>669</td>
<td>4124</td>
<td>0:00:01:653</td>
<td>1:58:21:048</td>
</tr>
<tr>
<td>lsm</td>
<td>468</td>
<td>8</td>
<td>11</td>
<td>168</td>
<td>2464</td>
<td>0:00:00:031</td>
<td>1:58:21:048</td>
</tr>
<tr>
<td>svchost</td>
<td>556</td>
<td>8</td>
<td>11</td>
<td>375</td>
<td>4080</td>
<td>0:00:04:820</td>
<td>1:58:20:330</td>
</tr>
<tr>
<td>svchost</td>
<td>632</td>
<td>8</td>
<td>11</td>
<td>385</td>
<td>5668</td>
<td>0:00:01:887</td>
<td>1:58:20:174</td>
</tr>
<tr>
<td>MsMpEng</td>
<td>680</td>
<td>8</td>
<td>18</td>
<td>371</td>
<td>105612</td>
<td>0:07:54:695</td>
<td>1:58:20:177</td>
</tr>
<tr>
<td>svchost</td>
<td>820</td>
<td>8</td>
<td>33</td>
<td>606</td>
<td>19044</td>
<td>0:00:00:904</td>
<td>1:58:19:862</td>
</tr>
<tr>
<td>svchost</td>
<td>868</td>
<td>8</td>
<td>24</td>
<td>432</td>
<td>6912</td>
<td>0:00:00:468</td>
<td>1:58:19:784</td>
</tr>
<tr>
<td>svchost</td>
<td>896</td>
<td>8</td>
<td>31</td>
<td>545</td>
<td>18004</td>
<td>0:00:01:419</td>
<td>1:58:19:722</td>
</tr>
<tr>
<td>svchost</td>
<td>952</td>
<td>8</td>
<td>75</td>
<td>1628</td>
<td>58120</td>
<td>0:04:47:057</td>
<td>1:58:19:287</td>
</tr>
<tr>
<td>svchost</td>
<td>1100</td>
<td>8</td>
<td>29</td>
<td>705</td>
<td>40132</td>
<td>0:00:02:308</td>
<td>1:58:04:839</td>
</tr>
<tr>
<td>spoolsv</td>
<td>1224</td>
<td>8</td>
<td>27</td>
<td>517</td>
<td>14760</td>
<td>0:00:00:421</td>
<td>1:58:03:154</td>
</tr>
<tr>
<td>svchost</td>
<td>1252</td>
<td>8</td>
<td>31</td>
<td>363</td>
<td>9348</td>
<td>0:00:00:670</td>
<td>1:58:03:030</td>
</tr>
<tr>
<td>eBSvc</td>
<td>1356</td>
<td>8</td>
<td>33</td>
<td>173</td>
<td>4360</td>
<td>0:00:00:062</td>
<td>1:58:02:530</td>
</tr>
<tr>
<td>mDNSv</td>
<td>1540</td>
<td>8</td>
<td>5</td>
<td>69</td>
<td>1192</td>
<td>0:00:00:031</td>
<td>1:58:01:996</td>
</tr>
<tr>
<td>AppleMobileDeviceService</td>
<td>1576</td>
<td>8</td>
<td>11</td>
<td>136</td>
<td>3252</td>
<td>0:00:00:062</td>
<td>1:58:00:799</td>
</tr>
<tr>
<td>mDNSResponder</td>
<td>1608</td>
<td>8</td>
<td>6</td>
<td>118</td>
<td>1940</td>
<td>0:00:00:530</td>
<td>1:58:00:440</td>
</tr>
</tbody>
</table>
consoles

Screen 0x186fd0 X:80 Y:300
Dump:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>f:

F:\>dir
Volume in drive F is PATRIOT
Volume Serial Number is 52E1-6FB3

Directory of F:\
04/10/2014 01:26 PM  268,435,456 ext3-256MB.img
04/10/2014 02:05 PM  5,162 vizdiff.pl
04/01/2014 08:19 AM  <DIR> IMAGES
04/10/2014 01:26 PM  268,435,456 msdos-256MB.img
04/10/2014 01:26 PM  268,435,456 ntfs-256MB.img
04/10/2014 01:26 PM  268,435,456 zeros-256MB.img
08/24/2014 09:38 AM  893,413 Xubuntu1404.zip
04/17/2014 02:11 PM  997 otr.private_key
07/22/2014 07:16 PM  957,349,888 xubuntu-14.04-desktop-amd64.iso
09/06/2014 05:21 PM  2,678,614,016 Windows Vista RTM x86.iso
03/03/2013 05:34 AM  <DIR> winpmem-1.4
06/25/2014 08:46 AM  3,181,248,512 WIN7.ISO
   10 File(s)  7,891,853,812 bytes
   2 Dir(s)  90,739,441,664 bytes free

F:\>cd winpmem-1.4

F:\winpmem-1.4>dir
Volume in drive F is PATRIOT
Volume Serial Number is 52E1-6FB3

Directory of F:\winpmem-1.4
$ python vol.py -f carberp.mem --profile=WinXPSP3x86 malfind
Volatility Foundation Volatility Framework 2.4
[snip]

Process: svchost.exe Pid: 992 Address: 0x9d0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x009d0000  b8 35 00 00 00 e9 8b d1 f3 7b 68 6c 02 00 00 e9 .5........{hl....
0x009d0010  94 63 f4 7b 8b ff 55 8b ec e9 6c 11 e4 7b 8b ff .c {...U...1...{
0x009d0020  55 8b ec e9 99 2e 84 76 8b ff 55 8b ec e9 74 60 U.......v...U....t`
0x009d0030  7f 76 8b ff 55 8b ec e9 8a e9 7f 76 8b ff 55 8b .v..U.......v..U.

0x9d0000  b835000000  MOV EAX, 0x35
0x9d0005  e98bd1f37b  JMP 0x7c90d195
0x9d000a  686c020000  PUSH DWORD 0x26c
0x9d000f  e99463f47b  JMP 0x7c9163a8
0x9d0014  8bff  MOV EDI, EDI
0x9d0016  55  PUSH EBP

Legitimate instructions
Simple Case Study: Memory Analysis vs. Spyware

Goal: Recover Credentials that Control Configuration Without RE
### Credit Card Statement

<table>
<thead>
<tr>
<th>Reference</th>
<th>Sold</th>
<th>Posted</th>
<th>Activity Since Last Statement</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>89XB773</td>
<td>12/17</td>
<td>Payment Thank You</td>
<td>-10.00</td>
<td></td>
</tr>
<tr>
<td>78XY667</td>
<td></td>
<td>1/10</td>
<td>Pizza Palace</td>
<td></td>
</tr>
<tr>
<td>34XP889</td>
<td></td>
<td>1/10</td>
<td>Small Town US</td>
<td>$24.53</td>
</tr>
<tr>
<td>23XY001</td>
<td>1/5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76X0E11</td>
<td></td>
<td>1/10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Account Number:** 234 567 8901
- **Name:** Suzy Student
- **Statement Date:** 1/15/2005
- **Credit Line:** $1,000.00
- **Credit Available:** $500.00
- **New Balance:** $1,000.00
- **Minimum Payment Due:** $30.00

### Finance Charge Summary

<table>
<thead>
<tr>
<th>Description</th>
<th>Periodic Rate</th>
<th>Annual Percentage Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchases</td>
<td>3%</td>
<td>36%</td>
</tr>
<tr>
<td>Advances</td>
<td>3%</td>
<td>36%</td>
</tr>
</tbody>
</table>

- **Current Amount Due:** $1,000.00
- **Amount Past Due:**
- **Amount Over Credit Line:**
- **Minimum Payment Due:** $30.00

Send Payment To:
PO Box 555
Anytown, US

**eBlaster $99.00**

Enter Password: [ ]

Copyright 2014-2017 by Golden G. Richard III (@nolaforensix)
Hotkey sequence brings up bare authentication dialog: <CTRL><SHIFT><ALT>-T.

This sequence can be changed. Can’t access configuration screen without the password.

Process is spawned to handle this dialog.

Process is terminated when dialog is dismissed.
Computer powered off, but eBlaster is persistent.

Clone and virtualize system to investigate.

dd image → raw2vmdk

VMWare virtual machine

Live forensics analysis
DumpIt window (dumps physical memory)
Goal: **quickly** reveal password w/o full reverse engineering effort.

Did developers leak the plaintext password in memory?

Enter sample password “VERYUNIQUE” in password dialog.

Don’t dismiss the “invalid password” dialog.

Dump memory in the VM while the dialog box is displayed.

YOUR-US67PI6LUV-20121017-214253.raw
# gstrings -a -t d YOUR-US67PI6LUV-20121017-214253.raw | grep VERYUNIQUE > strings.txt
# gstrings -a -t d -e l YOUR-US67PI6LUV-20121017-214253.raw | grep VERYUNIQUE >> strings.txt

467105092:VERYUNIQUE

Search memory dump for invalid password VERYUNIQUE

# python vol.py --profile=WinXPSP3x86 -f YOUR-US67PI6LUV-20121017-214253.raw strings -s strings.txt --output-file=stringslocation.txt -S

1bd77544 [3772:0012f544] VERYUNIQUE

Map location of invalid pw to owning process

PID of process whose address space includes string

# python vol.py --profile=WinXPSP3x86 -f YOUR-US67PI6LUV-20121017-214253.raw pslist

... 0x82c24bf0 xmlavipv.exe 3772 1592 2 78 0 0 2012-10-17 21:41:20

Verify that it’s the process that gets spawned when dialog opens

# python vol.py --profile=WinXPSP3x86 -f YOUR-US67PI6LUV-20121017-214253.raw -p 3772 memdump -D dumpdir

Dump memory of that process
Then run strings on dumped process virtual address space
Report Delivery Summary

Based on your current settings, eBlaster will send an Activity Report via email every 60 minutes to 'example.com'.

eBlaster will automatically forward Emails (including attachments), Chat / Instant Messages and Keyword Alerts to 'example.com'.

Activity Reports

- Activity Report Delivery: On/Off
- Send a Report: Every 60 Minutes
- Format Report as: HTML/Plain Text

Email

- Forward All Emails: On/Off
- Include Attachments: On/Off

Chat / Instant Message

- Forward All Chat/IMs: On/Off

Alerts

- Forward Keyword Alerts: On/Off
A Few Gaps

Userspace

GPUs, et al

Performance and Reliability

New OS Features
New OS Features

Windows 10 Virtualization Based Security

KARL: A new kernel on every boot...

Compressed RAM

Swap file on SSD

1+GB/s

1
Not present

Not present

0
4096

4GB
Compressed RAM
“Traditional” forensics: capture contents of storage devices, including the swap file.

Swap file contains RAM “overflow”—without this data, don’t have a complete view of system memory.

Memory forensics: capture RAM.

Acquiring both RAM and disk contents results in memory smearing, where the swap file may be (very) out of sync with the memory dump.
Forensically, Swap Files are a Mess

- Data trapped in unsanitized blocks allocated to swap file

```
void swap_crypt_ctx_initialize ( void ) {
    unsigned int i;
    if ( swap_crypt_ctx_initialized == FALSE ) {
        for ( i = 0; i < ( sizeof ( swap_crypt_key ) / sizeof ( swap_crypt_key[0] )); i++ ) {
            swap_crypt_key[i] = random();
        }
        aes_encrypt_key ( (const unsigned char *) swap_crypt_key ,
                          SWAP_CRYPT_AES_KEY_SIZE ,
                          &swap_crypt_ctx.encrypt);
        ...
        swap_crypt_ctx_initialized = TRUE ;
    }
}
```
All Evidence Sources: Compressed RAM

Mac OS X + Linux + Win10 set aside RAM and compress pages that would otherwise have been swapped due to memory pressure.
Mac OS X VM System

xnu-2422.1.72/osfmk/vm:
~73K lines of C

xnu-2422.1.72/osfmk/x86_64/WKdm*.s:
~1000 lines of 64-bit assembler for
WKdm_compress_new() /
WKdm_decompress_new()

Just vm_compressor /
vm_compressor_pager:
~4K lines of C + the assembler, above
Compressed RAM: Visual

Physical memory dump:

Have:

Want:
Compressed RAM: The Scoop

put the rose in my hair like the Andalusian girls used or shall I wear a red yes and how he kissed me under the Moorish wall and I thought well as well him as another and then I asked him with my eyes to ask again yes and then he asked me would I yes to say yes my mountain flower

You're reading. We're hiring.
https://www.flickr.com/jobs/

<html lang="en-us">
<head>
</head>
<body>

You're reading. We're hiring.
https://www.flickr.com/jobs/

</body>
</html>
Benchmarks (Sorry, Python)

WKdm compression / decompression ops/sec

All on 3.4GHz i7 iMac w/ 32GB RAM

Also believes this is unacceptable →
Reliability
Fuzzing Memory Forensics Frameworks

Fuzzing Harness

Mutation Set

FUSE Filesystem

Memory Image
Fuzzing

Requires root

Fuzziing Harness

Mutation Set

FUSE Filesystem

Memory Image

Copyright 2014-2017 by Golden G. Richard III (@nolaforensix)
## Volatility: Crashes

<table>
<thead>
<tr>
<th>Plugin Name</th>
<th>Programming Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_library_list</td>
<td>List enumeration code didn’t properly validate pointer to data structures before processing it</td>
</tr>
<tr>
<td>linux_dmesg</td>
<td>Did not validate that log structures referenced a valid page before attempting to process them</td>
</tr>
<tr>
<td>linux_arp</td>
<td>Integer overflow in bit shifting operation</td>
</tr>
<tr>
<td>mac_check_syscall</td>
<td>Crashed when system call table entries were not on a mapped page</td>
</tr>
</tbody>
</table>
## Volatility: Infinite Loops

<table>
<thead>
<tr>
<th>Plugin Name</th>
<th>Programming Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_bash</td>
<td>List enumeration code infinite looped when a mutation caused two list members to point to each other</td>
</tr>
<tr>
<td>linux_arp</td>
<td>List enumeration code infinite looped when a mutated list entry pointed to a previous entry</td>
</tr>
<tr>
<td>mac_lsmod</td>
<td>List enumeration code infinite looped when a mutated list entry pointed to a previous entry</td>
</tr>
<tr>
<td>mac_lsof</td>
<td>Nearly infinite loop when the variable specifying how many handles a process had opened was mutated to ~3 billion</td>
</tr>
</tbody>
</table>
## Volatility: Unusable Output

<table>
<thead>
<tr>
<th>Plugin Name</th>
<th>Programming Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_psaux</td>
<td>The <em>mm_struct</em> members that specify the start and end of the command line arguments were mutated to specify a size in the gigabytes</td>
</tr>
<tr>
<td>linux_psenv</td>
<td>Same as issue as <em>linux_psaux</em>, but for the members that specify the size of the process’ environment variables</td>
</tr>
<tr>
<td>mac_dyld_maps</td>
<td>List enumeration code infinite looped and the rendering code did not validate structure properly before reporting (printing)</td>
</tr>
<tr>
<td>mac_psaux</td>
<td>Same base issue as <em>linux_psaux</em></td>
</tr>
</tbody>
</table>
# Volatility: Resource Exhaustion

<table>
<thead>
<tr>
<th>Plugin Name</th>
<th>Programming Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>linux_procdump</td>
<td>Gaslight forced the plugin to attempt to create a 20GB+ file as the function used for extraction to disk, <em>write_elf_file</em>, did not properly validate file size metadata</td>
</tr>
<tr>
<td>linux_librarydump</td>
<td>This plugin relied on the same <em>write_elf_file</em> function when extracting shared libraries to disk</td>
</tr>
</tbody>
</table>
Userland Malware Detection
Crisis: Notorious Mac OS X Malware

- Kernel and user-level components
- Patches Activity Monitor to hide
- Code injection and Objective C pointer swizzling
- Takes screenshots, captures audio, video
- Snoops web browsing
- Intercepts and logs messaging
  - ...even if you use encryption protocols like OTR (Adium)
- Basically, all the stuff you're scared malware will do to you
Pointer Swizzling Detection on Mac OS X

- Better kernel-level defenses = renewed interest in userland malware
- Focus on Objective-C malware (e.g., Crisis)
- Complex runtime provides lots of opportunities
  - Add methods
  - Switch implementations
  - ...

```swift
swizzleByAddingIMP(
    @selector(webFrameLoadCommitted:),
    @selector(webFrameLoadCommittedHook:));
...
swizzleByAddingIMP(
    @selector(closeCurrentTab:),
    @selector(closeCurrentTabHook:));
...
swizzleByAddingIMP(
    @selector(setTitle:),
    @selector(setTitleHook:));
...```
Swizzling Detection

$ python vol.py -f memdump.raw mac_swizzled -p 1497

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>Class</th>
<th>Method</th>
<th>Method Address</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>dealloc</td>
<td>0x00007fffffff95ba9d7f</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>finalize</td>
<td>0x00007fffffff95ba9ead</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>image</td>
<td>0x00007fffffff95ba9f5c</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>isEnabled</td>
<td>0x00007fffffff95ba9d6e</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>hasMarkedText</td>
<td>0x00007fffffff95baa235</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>selectedRange</td>
<td>0x00007fffffff95ba2e5</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
<tr>
<td>kl</td>
<td>1497</td>
<td>NSInputManager</td>
<td>insertText:</td>
<td>0x00007fffffff95ba9fe0</td>
<td>/System/Library/.../Versions/C/AppKit</td>
</tr>
</tbody>
</table>

Warning signs:

- Implementation in different library than most other methods
- Address in non-file-backed memory region
Some Papers


golden@cct.lsu.edu / cct.lsu.edu/~golden

Twitter: @nolaforensix