DAMM: Differential Analysis of Malware in Memory

Dr. Vico Marziale
Managing Partner

OSDFC 2014
• Managing Partner @504ENSICS Labs in NOLA
• PhD in CS from UNO
  • Briefly taught security/crypto stuff
• Many hats:
  • R&D
  • Penetration testing
  • Malware analysis
  • Digital Forensics
• Contributor to/developer of: Registry Decoder, Scalpel, Spotlight Inspector, DAMM
• Co-organizer BSidesNOLA
• Tequila enthusiast
Malware Analysis: Hard!

• That is a problem. Do I get a cookie?

• Multiple types of analysis
  • Static (look at the binary)
  • Dynamic (run the binary and look)
  • Memory Analysis (a hybrid approach)

• Pros and cons re:
  • Time/expertise required
  • What malicious activity can be found
  • How malware can make life difficult

• Goal: Make memory analysis a bit easier. Cookie now?
Static Analysis

- **Tools:** disassemblers, decompilers, string/grep
- **Pros:**
  - Malware that’s not running can’t actively mess with you
    - Hiding processes
  - Or infect your network
  - Can see all environment based possible actions
- **Cons:**
  - Also can’t unpack itself, decrypt itself, or perform network activity (download files, C2 communications)
  - Disassembly/reversing is hard (takes time, expertise)
  - Can’t directly see effects on entire system
Dynamic Analysis

- Tools: debuggers, sandboxes, procmon, fakenet
- Pros:
  - Must be unpacked and decrypted to execute
  - Can see network activity
  - Less need for disassembly/reversing
  - Can potentially see entire running system
- Cons:
  - Peek-a-boo - it can (probably) see you
    - Whether debugger, host-based analysis, VM, or sandbox
  - Then lie to you (_EPROCESS linked lists ++)
  - And/or alter its behavior: sleep, exit, crash box, migrate via speaker/microphone (maybe ...)
  - And perform malicious activities
Memory Analysis

• A bit like each of the previous
• Run the malware (host, VM, sandbox)
  • Like dynamic analysis
• Wait a sec (min, hour, day)
  • Some malware is sleepy
• Make copy or copies of memory
  • Bit for bit copy (snapshot) of physical RAM
• Analyze snapshot(s)
  • Like static analysis
Memory Analysis

• Pros:
  • Malware unpacked/unencrypted in memory
  • Network activity occurs
  • Malware captured in snapshot can’t mess with you
  • We get to relax and look at the entire running state of the system

• Win, right?
Problems

• Tons of Stuff
  • Have a 256GB image of RAM. Now what? Volatility!

• Great: parses all the things
  • Processes, network connections, DLLs, modules, open handles to files, sockets, registry, mutexes, etc.

• Difficulty: parses all the things
  • 10s of processes, sockets, connection
  • Hundreds of DLLs, loaded modules, and services
  • Thousands of handles for
    • Files, sockets, mutants, registry keys
Problems

• Tons of Samples
  • Always more samples than analysis muscle
• What is malicious?
  • Most are stock Windows objects: might not need to focus initially on these
  • Or at least are made to look like stock Windows objects (lssass.exe): definitely need to look at these
• Automation the key (At least for triage)
• Many type of analysis are not easy to automate (IDA)
• Some types are easier (Cuckoo)
• How about for Volatility?
• It is extensible and open source (and fun)
DAMM Intro

- Differential Analysis of Malware in Memory
- FOSS tool built on Volatility
- Initially funded by DARPA Cyber Fast Track
- Python
- Command line
- Windows centric so far (but not for long)
- Duplicates Volatility output for many plugins
  - ~30 Volatility plugins combined into ~20 DAMM plugins
- Not so interesting
DAMM Intro (Cont.)

- Can analyze multiple copies of RAM
  - E.g., clean versus (suspected) infected
- And highlight differences between them
  - New drivers, processes, etc.
  - Changes in above
- SQLite backed
- Smart type-aware filtering
- Issue warning of ‘suspicious’ artifacts
- Multiple output formats
- Library-ized: libdamm (parse stuff into objects)
- Data reduction, expert domain knowledge, friendly output, performance
- Beta-ish!
Use Cases

• In virtual infrastructure
  • Snapshot RAM at each boot
  • See changes from boot to current
  • Or look further back in time

• Non-virtual environments
  • Keep Gold Standard disk image?
  • Do same for memory image!
    • Or generate as needed

• Malware analysis sandboxes
  • Configure to take before and after memory snapshots
Basic Usage

#python damm.py --profile WinXPSP2x86 -f mem.img -p processes

Lists interesting information about running and exited processes

Output clipped, also gives start time, exit time, invocation, number of threads and open handles, etc. (combines pslit, psscan, psxview, ...)

<table>
<thead>
<tr>
<th>offset</th>
<th>name</th>
<th>pid</th>
<th>ppid</th>
<th>prio</th>
<th>image_path_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25c8830</td>
<td>System</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0x225ada0</td>
<td>alg.exe</td>
<td>188</td>
<td>668</td>
<td>8</td>
<td>C:\WINDOWS\System32\alg.exe</td>
</tr>
<tr>
<td>0x2114938</td>
<td>ipconfig.exe</td>
<td>304</td>
<td>968</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0x2086978</td>
<td>TSVNCache.exe</td>
<td>324</td>
<td>1196</td>
<td>8</td>
<td>C:\Program Files\TortoiseSVN\bin\TSVNCache.exe</td>
</tr>
<tr>
<td>0x22df020</td>
<td>smss.exe</td>
<td>376</td>
<td>4</td>
<td>11</td>
<td>\SystemRoot\System32\smss.exe</td>
</tr>
</tbody>
</table>

Similar plugins exist for ~20 other types of objects: dlls, network connections ...
DAMM: Performance

User can opt to store results in SQLite db:

```
#python damm.py --profile WinXPSP2x86 –f mem.img –p processes –db mem.db
```

- Makes re-parsing instant
- Can easily be shipped to other investigators
- Or serve as an archive
- Db includes some simple metadata
  - No more need for memory snapshot (for plugin stuff)
  - No more need to specify profile
DAMM: DB Query

To use the db:
#python damm.py –p processes –db mem.db

To see some of what is stored in the db:
#python damm.py –db mem.db –q

- Profile: WinXPSP2x86
- Memimg: cridex.vmem
- Computername: ACCOUNTING12
- Plugins: processes dlls injection ...

Also all of the envars for the explorer process (systemroot++ for warns)
Question?

• How do we determine what is the bad?
• Idea: get a clean copy of RAM from same/similar machine
• Compare before and after to infer malicious activity
  • New running processes
  • New loaded modules
  • …
• How, though? Use diff and we’re done, right?
To Diff or Not to Diff?

• Have two memory snapshots
• Each has a set of objects
  • Processes, DLLs, network connections, drivers
• How do we determine:
  • What *uniquely identifies* an object? (PID? Name?)
  • Which objects exist in both copies?
  • Only in the infected?
  • In both but changed (or not)?
  • Do the changes matter?
What is a ‘Process’

• Our notion has set of attributes
  • Name
  • PID
  • PPID
  • Physical address
  • Start time
  • # handles, threads
  • ...

• Same process on same boot of same machine?
  • Physical offset, pid, ppid, name?

• Different machines?
• Plain diff is simply not going to work
DAMM: Differential Analysis

• Use two memory snapshots
  • Before infection (or known clean)
  • After infection (or suspected of infection)

• Select plugin(s)

• Parse a set of objects from each snapshot into dbs
  • Processes, DLLs, etc. from Volatility into objects
  • Using shims. Belch.

• Generate differences

• View only
  • New objects in the ‘after’ or infected snapshot
  • Objects in both, but have changed

• Unique ID defaults set for same boot of same machine
Differencing Example

#python damm.py –p processes –db infected.db –diff clean.db

<table>
<thead>
<tr>
<th>status</th>
<th>offset</th>
<th>name</th>
<th>pid</th>
<th>ppid</th>
<th>prio</th>
<th>threads</th>
<th>handles</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>0x217f650</td>
<td>wpabaln.exe</td>
<td>1184</td>
<td>624</td>
<td>8</td>
<td>1</td>
<td>58</td>
</tr>
<tr>
<td>New</td>
<td>0x2408a78</td>
<td>wuauclt.exe</td>
<td>1596</td>
<td>1008</td>
<td>8</td>
<td>7</td>
<td>172</td>
</tr>
<tr>
<td>New</td>
<td>0x2288a78</td>
<td>WORDPAD.EXE</td>
<td>320</td>
<td>1204</td>
<td>8</td>
<td>2</td>
<td>98</td>
</tr>
<tr>
<td>New</td>
<td>0x22d3c10</td>
<td>cmd.exe</td>
<td>972</td>
<td>1956</td>
<td>8</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>New</td>
<td>0x216d228</td>
<td>win32dd.exe</td>
<td>1120</td>
<td>972</td>
<td>8</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>Changed</td>
<td>0x223d6a0</td>
<td>VMWARESERVICE.E</td>
<td>1812</td>
<td>668</td>
<td>13</td>
<td>2-&gt;3</td>
<td>82-&gt;132</td>
</tr>
<tr>
<td>Changed</td>
<td>0x247bb28</td>
<td>EXPLORER.EXE</td>
<td>1956</td>
<td>1932</td>
<td>8</td>
<td>16</td>
<td>293-&gt;427</td>
</tr>
</tbody>
</table>
DAMM: Unique ID

• For processes, unique id defined as
  • pid, ppid, name, creation_time
• This so far works for same machine same boot
• What about when not in controlled sandbox environment?
  • Like a diff with clean image from another execution
• Change the unique id
  • name, image_path_name, command_line?
  • Accounts for binaries in wrong places, and normal duplicate names: svchost
Unique ID Example (1)

Here the dbs were generated from 2 different WinXPSP2 images from different machines:

```
#python damm.py –db infected.db –p processes –diff clean.db
```

<table>
<thead>
<tr>
<th>status</th>
<th>offset</th>
<th>name</th>
<th>pid</th>
<th>ppid</th>
<th>prio</th>
<th>threads</th>
<th>handles</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>0x4b5a980</td>
<td>VMwareUser.exe</td>
<td>452</td>
<td>1724</td>
<td>8</td>
<td>8</td>
<td>204</td>
</tr>
<tr>
<td>New</td>
<td>0x655fc88</td>
<td>VMUpgradeHelper</td>
<td>1788</td>
<td>676</td>
<td>8</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>New</td>
<td>0x6945da0</td>
<td>spoolsv.exe</td>
<td>1432</td>
<td>676</td>
<td>8</td>
<td>14</td>
<td>137</td>
</tr>
<tr>
<td>New</td>
<td>0x1122910</td>
<td>svchost.exe</td>
<td>1028</td>
<td>676</td>
<td>8</td>
<td>88</td>
<td>1395</td>
</tr>
</tbody>
</table>

Everything (except System) shows up as new. Not helpful.
Here also the dbs were generated from 2 different WinXPSP2 images from different machines:

```
#python damm.py –db infected.db –p processes –diff clean.db –u name image_path_name command_line
```

<table>
<thead>
<tr>
<th>status</th>
<th>offset</th>
<th>name</th>
<th>pid</th>
<th>ppid</th>
<th>prio</th>
<th>threads</th>
<th>handles</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>0x10c3da0</td>
<td>wuaucld.exe</td>
<td>1732</td>
<td>1028</td>
<td>8</td>
<td>7</td>
<td>178</td>
</tr>
<tr>
<td>New</td>
<td>0x69d5b28</td>
<td>vmtoolsd.exe</td>
<td>1668</td>
<td>676</td>
<td>8</td>
<td>5</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>0x1bcd0b8-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Changed</td>
<td>&gt;0x1214660</td>
<td>System</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>56-61</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>0x18b4648-</td>
<td></td>
<td>1080</td>
<td>692</td>
<td>8</td>
<td></td>
<td>1140-</td>
</tr>
<tr>
<td>Changed</td>
<td>&gt;0x1122910</td>
<td>svchost.exe</td>
<td>&gt;1028</td>
<td>&gt;676</td>
<td>8</td>
<td>66-88</td>
<td>&gt;1395</td>
</tr>
</tbody>
</table>

More useful output, we see changed processes again (13/28 new)
DAMM: Filtering

• Further reduce set of objects
• Filter on objects’ attribute value: pid 4242
  • Find all about some process
  • DAMM knows PIDs versus other integers
• String search: string evil.dll
  • Find all occurrences of a DLL name
  • DAMM knows which attributes to search
• Filtering can be based on exact matching or partial
#python damm.py --db infected.db --p X --filter pid:1180

## Processes

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>PID</th>
<th>PPID</th>
<th>PRIORITY</th>
<th>Image Path Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4a4b5f8</td>
<td>lanmanwrk.exe</td>
<td>1180</td>
<td>1060</td>
<td>8</td>
<td>C:\WINDOWS\System32\lanmanwrk.exe</td>
</tr>
</tbody>
</table>

## Privileges

<table>
<thead>
<tr>
<th>PID</th>
<th>Filename</th>
<th>Value</th>
<th>Privilege</th>
<th>Present</th>
<th>Enabled</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1180</td>
<td>lanmanwrk.exe</td>
<td>17</td>
<td>SeBackupPrivilege</td>
<td>TRUE</td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>1180</td>
<td>lanmanwrk.exe</td>
<td>23</td>
<td>SeChangeNotifyPrivilege</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>1180</td>
<td>lanmanwrk.exe</td>
<td>20</td>
<td>SeDebugPrivilege</td>
<td>TRUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1180</td>
<td>lanmanwrk.exe</td>
<td>10</td>
<td>SeLoadDriverPrivilege</td>
<td>TRUE</td>
<td>TRUE</td>
<td></td>
</tr>
</tbody>
</table>

## Handles

<table>
<thead>
<tr>
<th>Offset</th>
<th>PID</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80f5c260</td>
<td>1180</td>
<td>File</td>
<td>...\Documents and Settings\Administrator\Desktop</td>
</tr>
<tr>
<td>0xe1621ec0</td>
<td>1180</td>
<td>Key</td>
<td>...\WINDOWS\CURRENTVERSION\RUN</td>
</tr>
<tr>
<td>0xe1dd8a70</td>
<td>1180</td>
<td>Key</td>
<td>...\WINSOCK2\PARAMETERS\NAMESPACE_CATALOG5</td>
</tr>
<tr>
<td>0xe10f5188</td>
<td>1180</td>
<td>Key</td>
<td>...\WINSOCK2\PARAMETERS\PROTOCOL_CATALOG9</td>
</tr>
<tr>
<td>0xe1e96d30</td>
<td>1180</td>
<td>Key</td>
<td>...\WINDOWS\CURRENTVERSION\INTERNET SETTINGS</td>
</tr>
</tbody>
</table>

And so on for DLLs, network connections, etc...
DAMM: Warnings

Automatically search for suspicious objects:

- Processes running from temp directories
- DLLs loaded from temp directories
- PE headers in injectable memory pages
- For core Windows processes: correct priority, parent-child relationship, binary path
- Hidden processes, dlls
- Mangled filenames for important processes
- MFT pf entries for suspicious processes
- From the Volatility cheat sheet, The Book, “Know Your Windows Processes or Die Trying,” and elsewhere
Code injection:
  • services.exe (pid: 668) has PE header in injection.

Number of process instances:
  • lsass.exe (pid: 1928) has 3 instances. Only one instance should exist!

Proper parent/child process relationships:
  • lsass.exe (pid: 1928) parent process expected: winlogon.exe, actual: services.exe.

Boot time processes starting long after boot:
  • lsass.exe (pid: 868, "C:\WINDOWS\system32\lsass.exe") started 18703082.0 seconds after boot time which may be suspicious.

Process priority:
  • lsass.exe (pid: 1928) base priority expected: 9, actual: 8.

Process unlinking:
  • 1_doc_RCCData_61 (pid: 1336) may be a hidden process.

Prefetch entries for suspicious processes
  • File: [MFT FILE_NAME] WINDOWS\Prefetch\REG.EXE-0D2A95F7.pf is a prefetch entry for a suspicious process.

Mangled names:
  • winninit.exe (pid: 4792) is named suspiciously similarly to a Windows process: wininit.exe.
Output Formats

To further ease analysis, output (using db) can be:

- TSV (for Excel or whatever, use –tsv)
  - Greppable (if filtering doesn’t suffice, use --grepable)

```plaintext
dlls: process_id: 584  process_name: csrss.exe  dll_base: 0x75b50000  load_count: 0x3
dlls: process_id: 584  process_name: csrss.exe  dll_base: 0x75b60000  load_count: 0x2
dlls: process_id: 584  process_name: csrss.exe  dll_base: 0x77dd0000  load_count: 0x5
dlls: process_id: 584  process_name: csrss.exe  dll_base: 0x7e720000  load_count: 0x1
```

- Screen formatted (the default, use ‘less –S’ or equivalent)
Conclusion

• That’s about it for now.

• Next up?
  • More warnings
  • Full Windows support
  • Then maybe Linux and Mac
  • More funding?

• Lots of things to think about in light of the presentations from yesterday and today
Questions?

Dr. Vico Marziale
vico@504labs.com
@vicomarziale

https://github.com/504ensicsLabs/DAMM