Automatic malicious code extraction using Volatility Framework

Martin G. Korman
# id

uid=0(martin) gid=0(korman) groups=1(memory analysis), 2(malware analysis) 3(HD forensics)

Forensic investigator at Team8 Consulting
VolatilityBot?

VolatilityBot is an automation tool for researchers, that cuts most of the guesswork and manual tasks when performing memory analysis for malware analysis purposes.

VolatilityBot helps the investigator in the first steps of performing a memory analysis investigation.

Also, it helps with the first steps of malware analysis, in order to get the malicious code unpacked in memory.
What’s new in v2

- Rewrite of complete codebase (in Python 3.5), to improve performance and make the code more readable for contributors
- New heuristics feature
- An option to analyse automatically memory dumps, instead of malware samples (inspired by @attrc), helpful to do memory analysis at scale

http://volatility-labs.blogspot.co.il/2016/08/automating-detection-of-known-malware.html
Malicious Code – Where is it hiding?
New Processes

- New processes created by the Malware

```
<table>
<thead>
<tr>
<th>7cc500fe59ed5cd398e7e4c4...</th>
<th>2156</th>
</tr>
</thead>
<tbody>
<tr>
<td>optrust.exe</td>
<td>2168</td>
</tr>
</tbody>
</table>
```
Self Modifying Code

- Packers and Crypters have to unpack the malware in memory, in order to execute it.

<table>
<thead>
<tr>
<th>Name</th>
<th>Virtual Size</th>
<th>Virtual Address</th>
<th>Raw Size</th>
<th>Raw Address</th>
<th>Reloc Address</th>
<th>Linenums</th>
<th>Relocations</th>
<th>Linenumbers</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte[8]</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>CEBE946E</td>
<td>CEB911A9</td>
</tr>
<tr>
<td>lhlpk</td>
<td>0022E000</td>
<td>00001000</td>
<td>0028A00</td>
<td>00004000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00004000</td>
<td>00000000</td>
</tr>
<tr>
<td>akepu</td>
<td>00222000</td>
<td>00224000</td>
<td>0019000</td>
<td>0028E100</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>ntrvg</td>
<td>00112000</td>
<td>00251000</td>
<td>0001000</td>
<td>00280300</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>tobor</td>
<td>00000000</td>
<td>00253000</td>
<td>0001600</td>
<td>0028E500</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>eooan</td>
<td>00000400</td>
<td>0025B000</td>
<td>0000600</td>
<td>00284400</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>xymfmd</td>
<td>00000400</td>
<td>0025F000</td>
<td>0000600</td>
<td>00284400</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>jyacrd</td>
<td>00000400</td>
<td>00270000</td>
<td>0000600</td>
<td>00284600</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>vnidw</td>
<td>00001000</td>
<td>00277000</td>
<td>0003A00</td>
<td>00284600</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CE94C4F3</td>
</tr>
</tbody>
</table>
Remote Code Injection

- A malicious process writes code into the memory space of a target process and forces it to execute.
Hollow Process Code Injection

• A malicious process starts a new instance of a legitimate process (i.e. explorer.exe, svchost.exe) in suspended mode. (0x00000004 Flag)

• Before resuming it, the executable section(s) are freed and reallocated with malicious code.

![Image of assembly code with highlighted instruction]
Kernel Modules

- KMDs are usually part of a rootkit
- Usually they serve to hide malware evidence, make the malware harder to remove or obstruct the research process.
- “Advanced control and data flow hijacking techniques that leverage the lower layers of the OS architecture.”
API Hooks

- Hi-jacking the code flow of a legitimate Windows API call, in order to make it do something else, i.e. grab your POST request.
KEEP CALM AND Keep Guessing
VolatilityBot

- Automated
- Modular
- Extraction of various artifacts
Manager

- Can theoretically manage an unlimited quantity of machines.
- Tags - Multiple tags can be defined on execution
- Dynamic Tags - Some post processing modules add tags to the sample. i.e.: Code_Injection, Hooks_API
Machines

- Abstract model of a machine that has five basic functions:
  - Revert
  - Suspend
  - Start
  - Get Memory Path
  - Cleanup
Code Extractors

- Injected Code
- Kernel Modules
- New Processes
- Entire Address Spaces
- Hooks
Post Process Modules

- YARA (And YARA Semantic Analysis)
- Strings
- Basic Static PE analysis
- IDC file with imports from memory
- Fix PE header (sections, image base...)

Diagram:
1. Manager
2. Code Extractors
3. Post Processing
YARA Semantic Analysis

- Dynamic YARA rules
- Detect specific behaviors

```plaintext
$process_priv_esc = {{ FF 15 4D 00 7D 98 50 FF 15 4D 00 7D A3 [ - ] 68 string:SeDebugPrivilege [ - ] FF 15 4D 00 7D A6 [ - ] 50 [ - ] FF 15 4D 00 7D D2 AdjustTokenPrivileges
```
Heuristics

- Spawned processes of supposedly exploitable processes (browsers, office, etc…)
- Processes launched from suspicious paths
- Suspicious handles
- Code injection
Examples
Analysing a malware sample with VolatilityBot

- Execute
- Extract memory
- Code Extractors
- New processes
- Loaded KMDs
- Injected code
- Strings
- PE Static Analysis
- YARA Results
- Golden Image
- Report
- Dumps
Analysing a memory dump with VolatilityBot
Efficacy & Results
Virus Share Malware Subset

Total Samples: 3875
Samples with at least one successful dump: 3395 (88%)
New processes dumped: 3363 (86%)
Injected Code extractions: 992 (25%)
Kernel Modules dumped: 119 (0.03%)

88% Success Rate
## Malware Families Subset

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Samples</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>Samples with at least one successful dump</td>
<td>63</td>
<td>92%</td>
</tr>
<tr>
<td>Injected Code extractions</td>
<td>41</td>
<td>60%</td>
</tr>
<tr>
<td>New processes dumped</td>
<td>31</td>
<td>45%</td>
</tr>
<tr>
<td>Kernel Modules dumped</td>
<td>4</td>
<td>0.05%</td>
</tr>
</tbody>
</table>

**92% Success Rate**
Demo Time!
### Malware Sample

**Select**

```sql
select timestamp, sha256, ephash, status from sample_malwaresample where id = 59
```

<table>
<thead>
<tr>
<th>timestamp</th>
<th>sha256</th>
<th>ephash</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-10-04 15:12:28.743153</td>
<td>d13eb5014c76370f063ef9f39a133c0e9b490a...</td>
<td>3eb74a8360993144b7da204c2c4affbd326e57...</td>
<td>completed</td>
</tr>
</tbody>
</table>

**Select**

```sql
select id, sha256, ephash, process_name, binary_path, source from sample_dump where sample_id = 59
```

<table>
<thead>
<tr>
<th>id</th>
<th>sha256</th>
<th>ephash</th>
<th>process_name</th>
<th>binary_path</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>952dbbaee5eb5504a285f11de30623e2b3e7852...</td>
<td>e7c9e670fa8786e0a6a4111d1e651481612ec27ab...</td>
<td>explorer.exe</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
</tr>
<tr>
<td>2</td>
<td>4915223c7a17d3b6a09ccc0a0c0827e9c85e9...</td>
<td>383db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a752b649fe529e47999999da8af0d640855f1ee...</td>
<td>338db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>822324845c0351a4c0cc3355611b61ef1d1396f9...</td>
<td>338db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b419abb441df5f5494984a2dce0874bc5a0ed2b2...</td>
<td>338db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>b952700d79d8f319e09481b911f4e22176e2a...</td>
<td>338db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0fc0e98757d1c1e8f88bf1df5aadd87d6a6d70f6...</td>
<td>338db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2cdf5ac2b83e3aaed671c1012dd41ab48f7ade1...</td>
<td>338db0b18c1a465b0745397786591e19596838...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7050eb2b317ee7f3f0e9a87906cb2ca39f1e9c1...</td>
<td>e7c9e670fa8786e0a6a4111d1e651481612ec27ab...</td>
<td>explorer.exe</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>injected_code</td>
</tr>
<tr>
<td>10</td>
<td>adc677c3195b5f707d0767522bbab8b370ff77f4...</td>
<td>77eb07a22e0afe9f8eb0b50925a8f7d39231...</td>
<td>taskhost.exe</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>procdump</td>
</tr>
<tr>
<td>11</td>
<td>91484c118ded36f5484ec4ad8d9a02490950890...</td>
<td>87249794b7600da1a38ab6dbd524904f0593cd5a...</td>
<td>WmiPrvSE.exe</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>procdump</td>
</tr>
<tr>
<td>12</td>
<td>4bcb8722709d323a99aa55ad3b3bbbe68f188...</td>
<td>810f6109260da1f289ee4a57c14a37a6e20a...</td>
<td>/Users/Martin/Projects/volatility.exe/Store/d13eb5014c7637</td>
<td>procdump</td>
<td></td>
</tr>
</tbody>
</table>
Basic static analysis of unpacked code
Yara results
Extracted strings

"string": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; InfoPath.1)"
"string": "ObtainUserAgentString"
"string": "[main]\r\nsrvurls=%s;%s\r\nsrvdelay=%s\r\nsrvretry=%s\r\nbuidlid=%s\r\n"
"string": "%d,%d %04d sp%ld,%ld %s"
Memory Dump

I took an APT Memory and Malware Challenge from SANS, and tried to answer some of the questions, using VolatilityBot:

```
bash-3.2$ /anaconda/bin/python VolatilityBot.py -m --dump -f /Users/Martin/Downloads/APT.img
```
What is the Process ID of the rogue process on the system?

What is the name of the rogue file itself?
How is the malware achieving persistence on the system?

What is the filename of the file that is hiding the presence of the malware on the system?
What is the name of the ISP that hosts the network where the malware is communicating with?
What’s Next?

- Integration with Fakenet-NG
- Integration with VT (For IP addresses)
- Automatic scan with Clam of dumps
- Automated Dumping of injected shellcode
- Extraction of malware configurations
- Additional information extraction (URLs, Mutexes)
Caveats

- False positives
- Anti-Research mechanisms
Cool! Where can I get it?

- My blog: https://forensi.cz
- For any questions or bug reports, do not hesitate to contact me via Twitter @MartinKorman
Thanks for you time!