Automatic malicious code extraction using Volatility Framework

Martin G. Korman
What’s your take away?

An open source tool, that extracts malicious code from memory and prepares it for deeper static analysis.
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>Linenumber</th>
<th>Relocations</th>
<th>Linenumber</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte[3]</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Dword</td>
<td>Word</td>
<td>Word</td>
<td>Dword</td>
<td></td>
</tr>
<tr>
<td>lhlbk</td>
<td>0022E000</td>
<td>00001000</td>
<td>0002BA00</td>
<td>00000400</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>40000C40</td>
<td></td>
</tr>
<tr>
<td>akexpu</td>
<td>00322000</td>
<td>0022E000</td>
<td>00019000</td>
<td>0002BE10</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000C88E</td>
<td></td>
</tr>
<tr>
<td>ntmg</td>
<td>00012000</td>
<td>00021000</td>
<td>00001000</td>
<td>0000E500</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000AB91</td>
<td></td>
</tr>
<tr>
<td>tobos</td>
<td>00300800</td>
<td>00263000</td>
<td>00004600</td>
<td>00045E30</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000C306</td>
<td></td>
</tr>
<tr>
<td>ecpan</td>
<td>00004000</td>
<td>00268000</td>
<td>00000600</td>
<td>0004A400</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000C454</td>
<td></td>
</tr>
<tr>
<td>xyfmd</td>
<td>00004000</td>
<td>0026F000</td>
<td>00000600</td>
<td>0004AA00</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000C54F</td>
<td></td>
</tr>
<tr>
<td>jyacd</td>
<td>00004000</td>
<td>00273000</td>
<td>00000600</td>
<td>0004B000</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000C56D</td>
<td></td>
</tr>
<tr>
<td>vridw</td>
<td>00010000</td>
<td>00277000</td>
<td>00003A00</td>
<td>0004B600</td>
<td>00000000</td>
<td>0000</td>
<td>0000</td>
<td>0000CB84</td>
<td></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.
Kernel Modules

• 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” ¹

---

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

Code Extractors

Post Processing

PE
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
- Generation of IDC file with imports from memory
- Fix PE header (sections, image base...)

Diagram:
- Manager
  - Code Extractors
  - Post Processing
YARA Semantic Analysis

- Dynamic YARA rules
- Detect specific behaviors
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

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th>Percentage</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**
What’s Next?

- 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?

- BitBucket Repository: https://bitbucket.org/martink90/volatilitybot_public/overview
- Communicate with me, via Mail in order to get the source code (kormanmartin@gmail.com)
- Use my Web-Service: https://fightingmalware.com
- @MartinKorman
- blog.fightingmalware.com
Questions?

Thanks for you time!