RoAMer: Robust Automated Malware Unpacker

Thorsten Jenke
Daniel Plohmann
Elmar Padilla

2019-04-03
Motivation

- In-depth analysis is a core building block for understanding malware
- Most of today’s malware is packed or obfuscated
- Unpacking is a necessary first step for analysis
- Automation of unpacking is highly desirable
There have been numerous previous approaches

They are either...

- not well tested against real-world malware
- lacking in generality
- lacking in evasion resilience
- lacking in throughput
- not released for the community
Introducing the Robust Automated Malware Unpacker (RoAMer)

It is a new automated generic unpacker

We evaluated our approach with two diverse data sets
  - Malpedia
  - Malshare 2017
Development

- Basically my Master Thesis
- Supervised by Daniel Plohmann and later by Elmar Padilla
Presentation agenda

1. Introduction
2. Prerequisites
3. Methodology
4. Implementation
5. Evaluation
6. Conclusion
7. Release
Unpacked vs. Dumped

- **Unpacked**
  - As close as possible to the original payload prior to packing
  - Typically achieved by intercepting execution after unwrapping
  - Can be executed as-is

- **Dumped**
  - Extracting system’s memory segments containing the payload (also known as dumping)
  - In most cases cannot be run as-is
  - Only an approximation of the original malware
  - Initialized data fragments
(Dis-)Advantages

- Dumping is technically easier to achieve
- Static analysis often does not require a perfect reconstruction of the original
- Unpacked samples are easier to utilize in dynamic analysis
- Dumps contain run-time data, such as dynamic imports and decrypted strings
- There are approaches for reconstructing samples from memory dumps into an executable form
- Goal of RoAMer: enable static analysis by dumping malware
6 Types of packers defined by Ugarte-Pedrero [3]

- Type I: Simplest packer
- Type II: Multiple simple packers in a line
- Type III: Multiple simple packers in a tree
- Type IV: Payload triggers packers
- Type V: Payload code is mangled with packer code
- Type VI: Decrypt on demand and then encrypted again
Presentation agenda

1. Introduction
2. Prerequisites
3. Methodology
4. Implementation
5. Evaluation
6. Conclusion
7. Release
Heavily inspired by the methodology of real-world analysts:

1. Execute malware in sandbox environment
2. Focusing on suspicious behavior like startup of new processes, sudden changes in memory sizes...
3. Dump new and suspicious regions
4. Decide whether to continue investigation or commence static analysis on dumps
Idea

- Assumption: New memory regions have to be allocated for malware to run
- This makes the payload directly observable and therefore "dumpable"
Therefore, algorithm returns a set of dumps of suspicious regions

Desired target dump among libraries, heap-sections, etc.

Filters have to be set in place to find the desired dump
Limitations

- Userland-only
- Requires one point in time where the whole image is exposed in memory
- Only packers I through V comply to this
- Typical problems with native execution of malware (sleep, specific time, etc.)
Presentation agenda

1. Introduction
2. Prerequisites
3. Methodology
4. Implementation
5. Evaluation
6. Conclusion
7. Release
System Design

- Written in Python 2.7 for Windows 7 32/64bit
- One component residing on the host system and the other in client
- Host part controls VM and interface to the user
- Client part (agent) is responsible for unpacking
- Interaction and observation with the memory is done through the Windows API
Output Filters

1. PE-header whitelist filter
2. Eliminate non-executable regions that are not adjacent to an executable region
Adressing Evasion Techniques

- Debugger detection
- Fingerprinting VM
- User Interaction
- Evading all other techniques: NtTerminateProcess-Hook
Metrics

- Correctness
- Precision
- Speed
Correctness

- Is the desired dump among the output?
- Number of additional dumps does not factor in
- True, False, No-changes
Precision

- How many undesired dumps are among the output?
- The additional amount of undesired dumps may become too high
- The higher the precision the higher the quality of the output
The time that the methodology needs to unpack the malware
Proposed method involves unsupervised execution of malware for predefined time
Unfeasibility of algorithm grows with the amount of time needed
Therefore speed is the amount of time passed until the first correct output dump is observable.
Setup

  - Malpedia
    - Well curated malware corpus
    - Ground truth through manually unpacked/dumped reference samples
  - Malshare 2017
    - Every PE-file uploaded to Malshare in 2017
    - Contains also potentially goodware and not packed samples
    - No ground truth available
  - Each sample runs for 10 minutes with and without hook
Malpedia Correctness

- TLSH to compare manually and automatically dumped samples
- Yara signatures from Malpedia’s database to determine correctness
Malpedia Correctness

Figure: Malpedia correctness
No ground truth available
Comparison of original header vs. dumped header
Therefore not packed samples and goodware considered incorrect
Malshare Correctness

Number of runs:
- With hook:
  - True: 2,241
  - False: 414
  - No changes: 91
- Without hook:
  - True: 1,088
  - False: 461
  - No changes: 1,197
Malpedia Precision

<table>
<thead>
<tr>
<th></th>
<th>Malpedia w/o Hook</th>
<th>Malpedia w/ Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25%</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td>29.7</td>
<td>19.1</td>
</tr>
<tr>
<td>Median</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>75%</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>max</td>
<td>17703</td>
<td>6313</td>
</tr>
</tbody>
</table>

**Table:** Number of dumps for Malpedia
Table: Overhead size for Malpedia

<table>
<thead>
<tr>
<th></th>
<th>Malpedia w/o Hook</th>
<th>Malpedia w/ Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>25%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Average</td>
<td>217%</td>
<td>439%</td>
</tr>
<tr>
<td>Median</td>
<td>1%</td>
<td>29%</td>
</tr>
<tr>
<td>75%</td>
<td>33%</td>
<td>112%</td>
</tr>
<tr>
<td>max</td>
<td>41338%</td>
<td>91575%</td>
</tr>
</tbody>
</table>
## Malshare Precision

<table>
<thead>
<tr>
<th></th>
<th>Malshare w/o Hook</th>
<th>Malshare w/ Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>min</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>25%</strong></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>7.8</td>
<td>11.29</td>
</tr>
<tr>
<td><strong>Median</strong></td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td><strong>75%</strong></td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td><strong>max</strong></td>
<td>89</td>
<td>171</td>
</tr>
</tbody>
</table>

**Table**: Number of dumps for Malshare
## Malshare Precision

<table>
<thead>
<tr>
<th></th>
<th>Malshare w/o Hook</th>
<th>Malshare w/ Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>25%</td>
<td>1%</td>
<td>3%</td>
</tr>
<tr>
<td>Average</td>
<td>200%</td>
<td>230%</td>
</tr>
<tr>
<td>Median</td>
<td>5%</td>
<td>25%</td>
</tr>
<tr>
<td>75%</td>
<td>48%</td>
<td>75%</td>
</tr>
<tr>
<td>max</td>
<td>112250%</td>
<td>112230%</td>
</tr>
</tbody>
</table>

**Table:** Overhead size for Malshare
## Malpedia Speed

<table>
<thead>
<tr>
<th></th>
<th>Malpedia w/o Hook</th>
<th>Malpedia w/ Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>25%</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Average</td>
<td>21.59</td>
<td>17.81</td>
</tr>
<tr>
<td>Median</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>75%</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Max</td>
<td>641</td>
<td>626</td>
</tr>
</tbody>
</table>

*Table: Dump timing for Malshare in seconds*
### Malshare Speed

<table>
<thead>
<tr>
<th></th>
<th>Malshare w/o Hook</th>
<th>Malshare w/ Hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>25%</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Average</td>
<td>17.02</td>
<td>15.39</td>
</tr>
<tr>
<td>Median</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>75%</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>Max</td>
<td>596</td>
<td>563</td>
</tr>
</tbody>
</table>

**Table:** Dump timing for Malshare in seconds
Discussion

- Hook increases correctness and speed
- Hook decreases precision
- Tradeoff between correctness, speed, and precision
- Enable or disable the hook according to circumstances
Presentation agenda

1. Introduction
2. Prerequisites
3. Methodology
4. Implementation
5. Evaluation
6. Conclusion
7. Release
RoAMer utilizes a technique commonly used by malware analysts
Evaluating RoAMer against two datasets
With promising results
Good basis for future work
Future Work

- Unpacker based on introspection
- Post-processing of dumps
- Increasing compatibility with all packer classes
- Other ways to determine end of monitoring phase
References

Cutler, S.
Malshare.
http://malshare.com/.

Plohmann, D., Clauss, M., Enders, S., and Padilla, E.
Malpedia: A collaborative effort to inventorize the malware landscape.

Ugarte-Pedrero, X., Balzarotti, D., Santos, I., and Bringas, P. G.
Sok: deep packer inspection: a longitudinal study of the complexity of run-time packers.
Presentation agenda

1. Introduction
2. Prerequisites
3. Methodology
4. Implementation
5. Evaluation
6. Conclusion
7. Release
• Code is not in a good place, yet
• Please wait for the release on https://github.com/UrmelAusDemEis/RoAMer
• thorsten.jenke@fkie.fraunhofer.de
• Thank you for your kind attention.