Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem

Alessandro Mantovani (EURECOM), Simone Aonzo (UniGe), Xabier Ugarte Pedrero (CISCO), Alessio Merlo (UniGe), Davide Balzarotti (EURECOM)
Packing

Original File
- Original PE Header
- .text
- .data, .rsrc, .rdata...

Packed File
- PE Header
- Packed Section
- Decompression Stub

Unpacked File in Memory
- Original PE Header
- .text
- .data, .rsrc, .rdata...

Packer Process

Unpacking Routine
Scope / Packing Definition

(Our definition of) packing implies

- Original code present, but NOT in an executable form
- Real code recovered at run-time

(Our definition of) packing does NOT include

- JIT compilers
- Droppers
- Emulators (Themida)
- Shellcode
Packed or not packed: that is the question

- Fundamental in malware analysis
- Wrong classification ⇒
  - costly and time-consuming dynamic analysis trying to unpack the sample
  - pollute the datasets used in many malware analysis studies
  - even worse, EVASION
- Our (false) friend: the entropy
  - compressed/encrypted data has high entropy levels
Our Agenda

1. The propagation of low-entropy packed samples
2. The adopted schemes
3. Current tools/approaches vs. low-entropy packed malware
Dataset

Do malware authors use low-entropy schemes to evade entropy checks?

- 50,000 PortableExecutable files (excluding libraries and .Net applications)
- 2013 - 2019
- Classified as malicious by more than 20 antivirus engines
- Entropy $H < 7.0$
  - entire file [1]
  - each section [2]
  - overlay data


Packer Detector (1/5)

PC

Lists status
WL = []
WXL = []
Packer Detector ($\frac{2}{5}$)

Lists status

\[ WL = [ ] \]

\[ WXL = [ ] \]
Lists status

\[ WL = [(0x1234, 0x2000); (0x1234, 0x2001)] \]

\[ WXL = [] \]

After executing the current instruction the memory at 0x2000 will be written
Packer Detector (\(\frac{4}{5}\))

Lists status

\[ \text{WL} = [(0x1234, 0x2000); (0x1234, 0x2001)] \]

\[ \text{WXL} = [ ] \]

Other instructions not affecting the memory at 0x2000
Packer Detector (5/5)

Lists status

$WL = [(0x1234, 0x2000); (0x1234, 0x2001)]$

$WXL = [(0x1234, 0x2000)]$

... xor eax, eax

mov WORD PTR [0x2000], 0x9090

... 0x00009090

0x00002000 0x00002004
Packer Detector - False Negatives

- False Negatives -- packed samples detected as not packed
  - unexpected crash
  - virtual environment detection
  - missing dependencies
  - incorrect command line arguments

- We discarded the samples that did not exhibit a sufficient runtime behavior
  - did not invoke at least 10 disk or network-related syscalls
  - samples whose executed instructions did not span at least five memory pages

- $50.000 - 3.705 = 46.295$
Hidden high-entropy data

While packed with a high-entropy scheme, these samples evaded our set of filters

- Encrypted data, but the data was
  - not stored in any of the section
  - nor in the overlay area
- **11.6%** (5.386/46.295)
  - dominated by two families: *hematite* and *hworld*
- E.g., *hematite*
  - file infector
  - area created between the PE header and the first section
Packer Detector - Results

31.5% (14.583/46.295) ⇒ entropy alone is a very poor metric to select packed samples
Schemes Taxonomy w.r.t. Entropy

1. **Decreasing**
   - Byte Padding
   - Encoding

2. **Unchanged**
   - Transposition
   - Monoalphabetic Substitution

3. **Slightly Increasing**
   - Polyalphabetic Substitution
Scheme Classifier

Relies on the output of Packer Detector ⇒ Written and eXecuted List [WXL]

- Every packing scheme needs to follow the same steps while unpacking
  - locate and access the source buffer that contains the packed data
  - perform operations on such data
  - write the unpacked data in the destination buffer
- We use PANDA to perform deterministic record and replay of a sample
  - \( \langle PCx, AWy \rangle \in [WXL] \)
  - backward data-flow analysis to locate the source buffer
- Decision making based on the byte distribution of source and destination buffers
# Scheme Classifier - Results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Type</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Padding</td>
<td>-</td>
<td>8.0</td>
</tr>
<tr>
<td>Encoding</td>
<td>standard</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td>custom</td>
<td>0.5</td>
</tr>
<tr>
<td>Mono-alphabetic Substitution</td>
<td>XOR</td>
<td>29.8</td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td>5.2</td>
</tr>
<tr>
<td></td>
<td>ROL/ROR</td>
<td>0.5</td>
</tr>
<tr>
<td>Transposition</td>
<td>-</td>
<td>0.3</td>
</tr>
<tr>
<td>Poly-alphabetic Substitution</td>
<td>XOR</td>
<td>46.9</td>
</tr>
<tr>
<td></td>
<td>ADD</td>
<td>2.8</td>
</tr>
<tr>
<td>Unknown</td>
<td>-</td>
<td>2.1</td>
</tr>
</tbody>
</table>
Case Study: Custom Encoding (*Emotet*)

Two layers of packing

- The second layer uses a custom high-entropy encryption with an 8-bytes long key
- The first layer reduces the entropy from 7.63 to 6.57
- Custom encoding + byte padding
- Packed data and keys stored in the sections: “.rsrс” and “.rdata”
Signature and Rule-Based Packing Detection

- Detect It Easy (DIE)
  - signatures based on a scripting language
- PEiD
  - signatures only contain low-level byte patterns
- Manalyze
  - signatures
  - PE structure heuristics
    - unusual section names
    - sections WX
    - low number of imported functions
    - resources bigger than the file itself
    - sections with H > 7.0
Signature and Rule-Based Packing Detection - Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Manalyze (signatures)</th>
<th>Manalyze (heuristics)</th>
<th>PEiD</th>
<th>Manalyze Sig (\land) PEiD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packed</td>
<td>242 (1.7%)</td>
<td>8358 (57.3%)</td>
<td>386 (2.6%)</td>
<td>214 (1.5%)</td>
</tr>
<tr>
<td>Not Packed</td>
<td>2518 (9.6%)</td>
<td>6023 (22.9%)</td>
<td>3438 (13.1%)</td>
<td>2487 (9.4%)</td>
</tr>
<tr>
<td>Hidden H-E data</td>
<td>0 (0%)</td>
<td>14 (0.3%)</td>
<td>2 (0.1%)</td>
<td>0 (0%)</td>
</tr>
</tbody>
</table>

- DIE detects no well-known packer in our entire dataset
- PEiD and Manalyze generated a large number of false positives
  - detected the presence of packing more often in unpacked samples than in the packed group
- Manalyze alerts are based on sections names used by some off-the-shelf packers
  - why the malware authors used those names?
  - they could be fake clues used on purpose to deceive automated tools
ML Packing Detection

- 15 approaches deal with this problem (SOTA)
- Several features categories
  - PE structure, heuristics, opcodes, n-grams, statistics, entropy
- Features vector ($\mathbf{W}$): union of all features from previous studies
  - A separate features vector excluding the entropy ($\mathbf{\bar{W}}$)
- The most popular classifiers: SVM, RF, MLP
- Dataset: low entropy packed + not packed (~40K)
ML Packing Detection - Results

NO classifier was able to identify accurately low-entropy packed malware!

\[
\text{Err}_{\text{notPacked}} = \frac{|FP|}{|TeS_{\text{notPacked}}|}
\]

\[
\text{Err}_{\text{packed}} = \frac{|FN|}{|TeS_{\text{packed}}|}
\]

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Training-Testing</th>
<th>Considering H</th>
<th>Not Considering H</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>75%-25%</td>
<td>4.43%</td>
<td>4.12%</td>
</tr>
<tr>
<td></td>
<td>50%-50%</td>
<td>4.31%</td>
<td>3.97%</td>
</tr>
<tr>
<td></td>
<td>25%-75%</td>
<td>4.44%</td>
<td>4.11%</td>
</tr>
<tr>
<td>MLP</td>
<td>75%-25%</td>
<td>6.34%</td>
<td>5.86%</td>
</tr>
<tr>
<td></td>
<td>50%-50%</td>
<td>6.87%</td>
<td>6.24%</td>
</tr>
<tr>
<td></td>
<td>25%-75%</td>
<td>6.89%</td>
<td>6.33%</td>
</tr>
<tr>
<td>RF</td>
<td>75%-25%</td>
<td>0.20%</td>
<td>0.23%</td>
</tr>
<tr>
<td></td>
<td>50%-50%</td>
<td>0.18%</td>
<td>0.20%</td>
</tr>
<tr>
<td></td>
<td>25%-75%</td>
<td>0.21%</td>
<td>0.20%</td>
</tr>
</tbody>
</table>
Conclusions

- Low-entropy packing schemes are a real and widespread problem
- Existing static analysis techniques are unsuccessful against them
  - Entropy
  - Signature and Rule-Based
  - Machine Learning
- There is need for new solutions
- Low-entropy packing schemes must be considered in future experiments

-- Thank you for your attention --