Pattern Recognition Techniques for the Classification of Malware Packers

Li Sun – RMIT University
Steven Versteeg – CA Labs, Melbourne
Serdar Boztaş – RMIT University
Trevor Yann – HCL Australia

ACISP 2010
Sydney, Australia, 5-7th July
This research was supported and sponsored by CA Labs and an ARC linkage grant.

- Ms. Sun was a Ph.D. candidate sponsored by CA Labs.
- A very warm thank you to CA and HCL for allowing access to their Virus Zoo during this project.
- Some results from this work are being implemented in the HCL antivirus engine.
Overview of Presentation

• Malware Overview
• Packers and their characteristics
• Packer classification
• Related work by other researchers
• Our approach
• Experiments and results
• Conclusions and Further work
Malware

- Malware is a catch-all term for any kind of Malicious software including computer viruses, internet worms, trojan horses, exploits, droppers, rootkits, etc.

- The software is malicious if it exhibits unanticipated or undesired effects, caused by an agent intent on damage to a single computer, server, or computer network.

- Malware employs a wide variety of code obfuscation techniques in order to hinder reverse-engineering. As well as slowing initial analysis, obfuscation serves to make simple variants of existing malware difficult to recognize, we give two examples in the next slide:
Malware

- **Entry-point obscuring (EPO):** A technique for infecting programs through which a virus tries to hide its entry point in order to avoid detection. Instead of taking control and carrying out its actions as soon as the program is used or run, the virus allows it to work correctly for a while before the virus goes into action.

- **Polymorphism:** Polymorphic viruses can change its decryptor's physical form for every instance while retaining the same basic routines, thus the detection solution designed by AV researchers can be applied at a later stage, e.g., after decryption.
Malware

- **Metamorphism**: A more sophisticated obfuscation is metamorphism, where the code body of the virus is changed for every new generation and scan strings and range scanning are of no use as detection techniques.

- **Encryption /Packing**: Files, or groups of files, are encrypted/compressed into another file so that they take up less space and hinder signature based malware detection.

- The focus of this talk is packing since it is the most common obfuscation technique. We focus on the PE (Portable Executable) class of software in Win32 systems.
PE Packing

- A *packer* is an executable program that takes an executable file or dynamic link library (DLL), compresses and/or encrypts its contents and then packs it into a new executable file (Figure in following slide).
- The appearance of the malicious code is changed after packing due to the compression and/or encryption. Therefore, the packed file will thwart malware detection as no signature match will be found.
- The analysis and detection of malware can only be undertaken after the file is unpacked—this is our focus here.
PE packing

Dos MZ Header
PE Header
Section Table
Sections
.unpacker
.packed data
Dos MZ Header
PE Header
Section Table
Sections
(Malware payload)
Terminology

- A packed file contains:
  - a first part which forms the compressed and/or encrypted original executable file;
  - a second part which is an unpacking stub which can dynamically recover the original executable file on the fly.

- For a valid PE file, the the magic value must be set to the assembly code of `MZ'. This is the DOS signature of a PE file as shown in the figure in the previous slide.

- OEP is “original entry point” of the packed executable.

- A packed file has a flat or almost flat byte frequency distribution. This will be important in the rest of the talk.
Byte frequency distribution of a packed file
PE unpacking

Dos MZ Header
PE Header
Section Table
Sections

.unpacker
.packed data
Dos MZ Header
PE Header
Section Table
Sections
(Malware payload)

...
Sophisticated packers

- Anti-unpacking tricks
  - Anti-dumping
  - Anti-debugging
  - Anti-emulating
  - Anti-intercepting

- The number of new packers and new variants of existing packers is increasing as time goes on
Packer classification

- Detect and identify the packer

- This will allow AV (Anti-virus) researchers to easily and correctly unpack the file and then…

- … retrieve original payload for further malware detection and analysis.
Traditional signature based packer classification (1)

- Based on byte signature match
- A unique signature for each specific version of a packer

**[UPX 0.70]**
signature = 60 E8 00 00 00 00 58 83 E8 3D
ep_only = true

**[UPX 0.72]**
signature = 60 E8 00 00 00 00 00 83 CD FF 31 DB 5E
ep_only = true
Traditional packer classification (2)

- Advantages:
  - Effective at detecting known packers

- Disadvantages:
  - Can’t cope with the sharp increase in new packers and the number of different variants of each packer
  - Expensive
    - Needs to be performed accurately by AV experts
    - Difficult, costly and time-consuming
Our approach

- Automatic packer classification system
Pattern recognition techniques

- Recognize a particular class from a measurement vector
  - Patterns with similar appearance tend to cluster together

- Feature extraction
  - Common features (patterns) among a set of objects

- Classification
  - Training set
  - Testing set
  - Classification model
Related work

- Pattern recognition hasn’t been used for the classification of the family of malware packers
- Perdisci et al.
  - classified packed vs non-packed files
- Most attempts of using pattern recognition have focused on malware detection
  - Tesauro et al.
  - Schultz et al.
  - The Malicious Executable Classification System (MECS)
  - Muazzam
Feature extraction (1)

- Packers’ randomness profiles were extracted
Feature extraction (2)

- We have used a refined local randomness test which comprises of:
  - Sliding window algorithm
  - Trunk pruning
  - No overlapping windows
  - Optimal parameters have been determined empirically
    - Window size = 32
    - Skip size = 32
    - Pruning size = 50
Feature extraction (3)

- Sliding window algorithm
  - This affords us to focus on local randomness
  - No overlapping windows were used—this was due to performance as parameters were varied and optimal parameters were determined empirically:
    - Window size = 32
    - Skip size = 32 [thus, no overlapping]
Feature extraction (4)

- Trunk pruning
  - Retain low randomness values at the beginning and the end of the file
  - Pruning size $n = 50$
    - $n$ is too small
      - Quick to build the classification model and classify the file
      - May not be sufficient information to distinguish between packers
    - $n$ is too large
      - Slow classification process
      - Introduce information noise
Classification

- Weka 3.6.0 package from Waikato University, NZ was used
- Four fast classifiers were tested
  - Naïve Bayes (NB)
  - Sequential Minimal Optimization (SMO)
  - k-Nearest Neighbor (kNN)
  - Best-first Decision Tree (BFTree)
- 10-fold cross validation was performed (as explained later)
Naïve Bayes (NB) 1

Based on the Bayes rule

- For a feature vector \( x \) with \( n \) attributes \( x = x_1, x_2, \ldots, x_n \) and a class variable \( y_j \),
- Let \( P(x \mid y_j) \) be the class-conditional probability for the feature vector \( x \) whose distribution depends on the class \( y_j \),
- \( P(y_j \mid x) \), the posterior probability that feature vector belongs to class can be computed as:

\[
P(y_j \mid x) = \frac{P(x \mid y_j)P(y_j)}{P(x)}
\]
Naïve Bayes (NB) 2

- Applies “naïve” conditional independence assumptions which states that all \( n \) features \( x_1, x_2, \ldots, x_n \) of the feature vector \( x \) are all conditionally independent of one another, given \( y_j \).

\[
P (x \mid y_j) = P (x_1 \ldots x_n \mid y_j) = \prod_{i=1}^{n} P (x_i \mid y_j)
\]

\[
P (y_j \mid x) = \frac{P (y_j) \prod_{i=1}^{n} P (x_i \mid y_j)}{P (x)}
\]
Sequential Minimal Optimization (SMO)

Based on the Support Vector Machine (SVM) algorithm
- SVM constructs an optimal hyperplane that separates a set of one class instances from a set of other class instances and maximizes the margin between the two data sets
Sequential Minimal Optimization (SMO) 2

- SMO decomposes the large QP (quadratic programming) problem of SVM into QP sub-problems.
  - Avoids using an entire numerical QP as an inner loop
  - Requires no extra matrix storage
    - The amount of memory required is linear in training set size
    - Can handle very large training sets
k-Nearest Neighbour (k-NN)

- One of the simplest machine learning algorithms – it is very fast and efficient.
- Training files that have the shortest distance to the test file are used to classify the test file by assigning it to the majority class within k-nearest neighbouring training files.
Best-first Decision Tree (BFTree)

- A decision tree that maps from attributes of an item to conclusions about its target class.
- Top down strategy is applied:
  - The “Best node”—which maximally reduces the impurity among all nodes available for splitting—is firstly added to the tree
  - This partitioning of the feature space is recursively executed until all nodes are non-overlapping or a specific number of expansions is reached.
  - In the latter case, pruning methods are used to decrease the noise and variability in the data and therefore to achieve better performance.
k-fold cross validation

- The whole data set is randomly partitioned into k equal-size subsets $D_1, D_2, \ldots, D_k$.
- A total of k runs of validation are performed.
- Each example is used as a test sample exactly once. i.e., for each run,
  - One subset is used as a test set to assess the performance of a fully-trained classifier, e.g., $D_1$.
  - The other $k-1$ subsets are used as a training set to train the classifier, e.g., $D_2, D_3, \ldots, D_k$.
- In this paper, we have used $k = 10$. 
Experimental data sets

- **Malware sample data set**
  - Real malware samples

- **Mixed sample data set**
  - Real malware samples
  - Packed clean files
Real malware samples

- All malware samples downloaded by CA/HCL over January and February in 2009 and stored in their “Virus Zoo”.
- Packer labelling performed using AV scanners
  - CA
  - Microsoft
  - Kaspersky
  - PEiD
- Total of 103,392 packed files were used in the experiment
Packed clean files

• Why?
  ◦ Include most sophisticated packed files in the data set
  ◦ Not enough real malware samples could be used
    • Most samples contain conflicting packer information provided by different AV scanners

• How?
  ◦ Clean files: 117 UnxUtils binaries
  ◦ Packer: Themida
    • 6 different versions of Themida are used.
Malware sample data set

- Two selection constraints:
  - Only confirmed cases are used.
    - Packer labelling is 100% correct.
  - Only packers with a sufficient number of confirmed cases are chosen. Thus, $N > 100$

<table>
<thead>
<tr>
<th>Packer</th>
<th>Versions</th>
<th>Total Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSG</td>
<td>1.33 and 2.0</td>
<td>5,105</td>
</tr>
<tr>
<td>NSPACK</td>
<td>2.x, 2.9, 3.4, 3.5, 3.6 and 3.7</td>
<td>256</td>
</tr>
<tr>
<td>PECOMPACT</td>
<td>2.xx</td>
<td>1,058</td>
</tr>
<tr>
<td>PETITE</td>
<td>2.1 and 2.2</td>
<td>152</td>
</tr>
<tr>
<td>UPACK</td>
<td>0.2x-0.3x</td>
<td>834</td>
</tr>
<tr>
<td>UPX</td>
<td>UPX, UPX(LZMA), UPX(Delphi), 2.90, 2.92(LZMA), 2.93 and 3.00</td>
<td>9,931</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>17,336</td>
</tr>
</tbody>
</table>
**Mixed sample data set**

- Packed clean files and malware samples.
- Contains sophisticated packers (in red).

<table>
<thead>
<tr>
<th>Packer</th>
<th>Versions</th>
<th>Total Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASPROTECT</td>
<td>unknown, 1.2 and 1.23</td>
<td>205</td>
</tr>
<tr>
<td>FSG</td>
<td>1.33 and 2.0</td>
<td>5,105</td>
</tr>
<tr>
<td>MEW</td>
<td>11 and 11 SE 1.2</td>
<td>261</td>
</tr>
<tr>
<td>NSPACK</td>
<td>2.x, 2.9, 3.4, 3.5, 3.6 and 3.7</td>
<td>256</td>
</tr>
<tr>
<td>PECOMPACT</td>
<td>2.xx</td>
<td>1,058</td>
</tr>
<tr>
<td>PETITE</td>
<td>2.1 and 2.2</td>
<td>152</td>
</tr>
<tr>
<td>THEMIDA</td>
<td>v1.8.0.0 with 6 option sets</td>
<td>117</td>
</tr>
<tr>
<td>UPACK</td>
<td>0.2x-0.3x</td>
<td>834</td>
</tr>
<tr>
<td>UPX</td>
<td>UPX, UPX(LZMA), UPX(Delphi), 2.90, 2.92(LZMA), 2.93 and 3.00</td>
<td>9,931</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>17,919</strong></td>
</tr>
</tbody>
</table>
Evaluation metrics (I)

- $TP$ – the true positives
- $FP$ – the false positives
- $TN$ – the true negatives
- $FN$ – the false negatives
- $P$ – the total number of positive records
- $N$ – the total number of negative records
Evaluation metrics (2)

- **True positive rate**
  \[ TP_{rate} = \frac{TP}{P} = \frac{TP}{TP + FN} \]

- **False positive rate**
  \[ FP_{rate} = \frac{FP}{N} = \frac{FP}{FP + TN} \]

- **Precision**
  \[ Precision = \frac{TP}{TP + FP} \]

- **Recall**
  \[ Recall = \frac{TP}{TP + FN} \]

- **Model building time**
Results obtained from the malware sample data set

- Randomness profile
  - Window size = 32
  - Feature vector size = 50

<table>
<thead>
<tr>
<th>Classifier</th>
<th>TP rate</th>
<th>FP rate</th>
<th>Model building time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayes.NaiveBayes</td>
<td>93.9%</td>
<td>1.1%</td>
<td>0.91</td>
</tr>
<tr>
<td>Functions.SMO</td>
<td>98.9%</td>
<td>0.8%</td>
<td>72.11</td>
</tr>
<tr>
<td>Lazy.kNN(k=1)</td>
<td>99.6%</td>
<td>0.1%</td>
<td>0.02</td>
</tr>
<tr>
<td>Trees.BFTree</td>
<td>99.3%</td>
<td>0.5%</td>
<td>16.45</td>
</tr>
</tbody>
</table>
Results of the mixed sample data set (1)

- kNN classifier with $k = 1$
- Feature vector size = 50
- The classification system is robust.
  - Packers with different sophistication levels can be classified.
  - Different variants of each packer can be classified into same family.
## Results of the mixed sample data set (2)

<table>
<thead>
<tr>
<th>Packer</th>
<th>TP rate</th>
<th>FP rate</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASPROTECT</td>
<td>92.7%</td>
<td>0.1%</td>
<td>95.5%</td>
<td>92.7%</td>
</tr>
<tr>
<td>FSG</td>
<td>99.9%</td>
<td>0.0%</td>
<td>99.9%</td>
<td>99.9%</td>
</tr>
<tr>
<td>MEW</td>
<td>99.6%</td>
<td>0.0%</td>
<td>100.0%</td>
<td>99.6%</td>
</tr>
<tr>
<td>NSPACK</td>
<td>91.0%</td>
<td>0.1%</td>
<td>90.7%</td>
<td>91.0%</td>
</tr>
<tr>
<td>PECOMPACT</td>
<td>98.5%</td>
<td>0.1%</td>
<td>98.2%</td>
<td>98.5%</td>
</tr>
<tr>
<td>PETITE</td>
<td>98.0%</td>
<td>0.0%</td>
<td>98.7%</td>
<td>98.0%</td>
</tr>
<tr>
<td>THEMIDA</td>
<td>92.3%</td>
<td>0.1%</td>
<td>86.4%</td>
<td>92.3%</td>
</tr>
<tr>
<td>UPACK</td>
<td>100.0%</td>
<td>0.0%</td>
<td>99.4%</td>
<td>100.0%</td>
</tr>
<tr>
<td>UPX</td>
<td>99.7%</td>
<td>0.3%</td>
<td>99.8%</td>
<td>99.7%</td>
</tr>
<tr>
<td><strong>Weighted Avg</strong></td>
<td><strong>99.4%</strong></td>
<td><strong>0.2%</strong></td>
<td><strong>99.5%</strong></td>
<td><strong>99.4%</strong></td>
</tr>
</tbody>
</table>
A novel automatic packer classification system was introduced and tested.

It gives very high accuracy on real malware:
- $\text{TPrate} = 99.6\%$
- $\text{FPrate} = 0.1\%$

Thus the *randomness profile* is a very strong feature for packer classification.
Further work

• Explore different statistical classifiers
  ◦ Multi-classifier algorithms
  ◦ Rank the output of different classifiers

• Enhance the feature extraction algorithm
  ◦ Important attributes
  ◦ Develop new effective pruning strategies
Thanks

- Any questions?