MALWARE DETECTION
based on
MACHINE LEARNING

Application and practice of machine learning in anti-malware

Ye Chao
Beijing Rising
Experience
2012

- x86 Instruction Flow based Predictor
  - OBSOLETE!
- PDF Exploit Predictor
  - OBSOLETE!
2012

Malware Predictor based on Decision-Tree

For Windows PE files:
- Millions of training samples
- Features extracted from file structure
- OBSOLETED!
Min-Hash & LSH based Clustering

find similar historical samples quickly and fall into one cluster
always select the latest sample to represent the cluster

270 millions strings filter

1 million+ cluster
2016

RDM+
malware predictor based on Random-Forest

For Windows PE
Tens of millions of training samples
Features are extracted from file structure/content/analysis
Use the Random Forest
RDM+
- A cautious predictor for malware detection
- It relies on file structure and part of the content
- It doesn't look so smart, but it improves through high frequency learning.
Feature Engineering
It is often said that

“In the application of machine learning, the feature engineering determines the upper limit of the model and algorithm performance.”
4778-D
Features Array
For RDM+

describes a file from multiple aspects from file content and file analysis results
Program Structure and Properties

Section Table Analysis

Entropy
'Size' Fields
Compiler

Relative Position of Important Data

......
Import/Export Symbol Names

Embody the intent of the program

An algorithm called IMPHASH is widely used in malware classification

Hash Trick

there is no need to create an encoding for each name
count the names by name hash
1024 slots  For IMPORT names

1024 slots  For EXPORT names

\[ \text{hash(\text{CreateFileA}) \% 1024} \]

\[ \text{hash(\text{setGlobalCallBack}) \% 1024} \]
In the obfuscated code, both the immediate number and the register are heavily used.

Frequently used instructions are grouped, others are completely reserved.
Strings in Section-Tables/Resources/Signature

Use "Alnum" table

“Micorsoft Windows ”

<table>
<thead>
<tr>
<th>M</th>
<th>i</th>
<th>c</th>
<th>r</th>
<th>o</th>
<th>s</th>
<th>f</th>
<th>t</th>
<th>W</th>
<th>n</th>
<th>d</th>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Features from Analysis
Insert many **fake API calls** in code to avoid the detection of some antivirus software, such as: Injector, Loader, Kryptik, XPACK, Crypter
The program is compiled by the ordinary compiler, but there is a lot of high entropy data in the code. After execution, the data is decoded into code and executed, such as: Injector, Loader, Kryptik, Crypter.
Symbols distribution is sparse in clean program

Symbols is densely distributed in some malware
The code between the first symbol and the last symbol almost fills the entire code section.

Very little code between the first and last symbols in some malware.
Features List

**ISRR**: imported symbols referenced ratio.
**ISCR**: imported symbols invoked ratio.
**ILRR**: imported libraries referenced ratio.
**ISDD(Max/Min)**: the density of symbols distribution in file.
**RPOS1**: the offset of first symbol divided by the section size.
**EDCR**: the compression rate of the executable data in program.
**IBR**: the ratio of branch instructions to total instructions (200).
**IDR**: to measure whether an instruction can be statically tracked.
**DER**: how many export symbols are in the data segment.
**BSR**: the ratio of BSS section size to image size.
**MSGR**: the ratio of the maximum size between two symbols and the code section size.
Model Training and Combination
Training Samples Set

20 million samples
remove duplicate samples
cluster filtering
100 million malware & clean files

~700G
actual number of bytes
Algorithm Selection

SVM
- Not suitable for a large number of samples
- Unable to complete training

Random-Forest
- Good effect on training set
- Key features can be found
- The training process is long

Decision-Tree
- Under-fitting
- The output is too simple to concatenate
Model Combination

4778-D
100 Trees in forest
Takes 120+ Hours

Unable to meet the hourly update

Model for Prediction
Model for dimensionality reduction
Model for Dimensionality Reduction

4778-D input
100-D output
Dimensionality reduction tool
Updated every few months

Model for Prediction

100-D input
100-D output
Prediction tool
Hourly update

After dimensionality reduction, the training difficulty is greatly reduced.
Prediction Model Training

Basic Samples & Latest Samples

**BS:** A set of historical samples after filtering and dimensionality reduction

+ 

**LS:** Recent major malware and clean files set, includes FPs

= 

5 million samples
covering about 50 million files
Prediction Model Training Time

0.78 hour

✔ Hourly update ✔ Model fine-tuning
Mitigating false positives
Missing malware is better than false positives!
How do we do that?

Choosing the right algorithm
In order to mitigate the false positives, we think that over-fitting is the advantage.

Masking false positives using hash value of features
In a production environment, the key-value database is used to mask false positives before predictions.

Carefully selected training samples
Select the right malware files and more clean files into the training set.
The cloud service
Compensating for model defects
Random-Forest cause the "model explosion" problem, making the model unsuitable for distribution to the host.

Requires high frequency updates
One is to maintain the most timely training and update, the second is to maintain timely false positives removal.
Operation Process
Performance
in the 1st month

80~90% Positives
~0.2% FPs

60~70% Positives
<0.1% FPs

after 3 months
In the 1st month

<table>
<thead>
<tr>
<th></th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injector/Kryptik</td>
<td>&gt;93%</td>
</tr>
<tr>
<td>Zbot</td>
<td>&gt;90%</td>
</tr>
<tr>
<td>Ransomware</td>
<td>&gt;92%</td>
</tr>
<tr>
<td>Detection</td>
<td>Details</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>AegisLab</td>
<td>Troj.W32.Gen.IIMFt</td>
</tr>
<tr>
<td>AVG</td>
<td>MSIL.Dropper-BE [Drp]</td>
</tr>
<tr>
<td>AVware</td>
<td>Virtool.MSIL.Injector.b (v)</td>
</tr>
<tr>
<td>CrowdStrike Falcon</td>
<td>malicious_confidence_100% (D)</td>
</tr>
<tr>
<td>Cyren</td>
<td>W32/MSIL_Troj.Rgen!Eldorado</td>
</tr>
<tr>
<td>Endgame</td>
<td>malicious (high confidence)</td>
</tr>
<tr>
<td>F-Prot</td>
<td>W32/MSIL_Troj.Rgen!Eldorado</td>
</tr>
<tr>
<td>Ikarus</td>
<td>Virtool.MSIL</td>
</tr>
<tr>
<td>McAfee-GW-Edition</td>
<td>BehavesLike.Win32.PUPXAG.gc</td>
</tr>
<tr>
<td>Qihoo-360</td>
<td>HEUR/QVM03.0.B65E.Malware.Gen</td>
</tr>
<tr>
<td>SentinelOne</td>
<td>static engine - malicious</td>
</tr>
<tr>
<td>VIPRE</td>
<td>Virtool.MSIL.Injector.b (v)</td>
</tr>
<tr>
<td>Ad-Aware</td>
<td>Clean</td>
</tr>
</tbody>
</table>

- Rising: Malware.Heuristic[ET]99% (RDM::zmIRtazoRc6Gz3X3/t05ZBUWt...
### 17 engines detected this file

**SHA-256**: `1ce06611080f4a1c0ba5f4da553e5fd181480163bc57876c7e096e3af022b708

**File name**: `notepad.exe.exe`

**File size**: `1.97 MB`

**Last analysis**: `2017-10-24 03:48:36 UTC`

<table>
<thead>
<tr>
<th>Detection</th>
<th>Details</th>
<th>Community</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avira</td>
<td>DR/Autoit.Gen2</td>
<td>W32.DropperZbotS.Trojan</td>
</tr>
<tr>
<td>CMC</td>
<td>Trojan-Spy.Win32.2bot:O</td>
<td>CrowdStrike Falcon</td>
</tr>
<tr>
<td>Cylance</td>
<td>Unsafe</td>
<td>eGambit</td>
</tr>
<tr>
<td>Endgame</td>
<td>malicious (high confidence)</td>
<td>ESET-NOD32</td>
</tr>
<tr>
<td>Fortinet</td>
<td>W32/Injector.LKtr</td>
<td>Kaspersky</td>
</tr>
<tr>
<td>Rising</td>
<td>Malware.Heuristic</td>
<td>ET #94%</td>
</tr>
<tr>
<td>Sophos ML</td>
<td>heuristic</td>
<td>TheHacker</td>
</tr>
<tr>
<td>ZoneAlarm</td>
<td>HEUR:Trojan.Win32.Generic</td>
<td>Ad-Aware</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clean</td>
</tr>
</tbody>
</table>
Other File Formats
### Different Formats vs. Different Features Engineering

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWF EXPLOIT</td>
<td>Features are extracted from flash structure and 3-grams of strings in ABC. Recent 30-Day performance: 520/563 ~ 92%, defeated almost all EXP-KITs.</td>
</tr>
<tr>
<td>Obfuscated Script</td>
<td>After special normalization, extract script skeleton features. It is still being improved because it often conflicts with '*.min.js'.</td>
</tr>
<tr>
<td>PDF EXPLOIT</td>
<td>Features come from PDF keywords and embedded JS. About 88% of PDF exploits/phishing can be detected.</td>
</tr>
</tbody>
</table>
Conclusion
• AI/ML can improve the productivity of all aspects of anti-malware.

• The goal of using ML needs to be clear.

• In our application, the feature engineering directly affects the final effect.

• It's important to mitigate false positives.
Continue To Challenge

Try to create a low-dimensional RDM+

More Feature Engineering

Behavior sequence + LSTM

Understanding API Calls

and so on
THANK YOU