Feature Set Selection in Data Mining Techniques for Unknown Virus Detection – A Comparison Study

Jianyong Dai
Ratan Guha
Joohan Lee
School of Electrical Engineering and Computer Science
University of Central Florida

Outline

- Problem Statement
- Simple Heuristics
- N-Gram
- Static Instruction Sequences Approach
- Dynamic Instruction Sequences Approach
- Experiments and Results
- Conclusion
Problem Statement

- Detecting known viruses
  - Virus signature
  - New virus coming out everyday
- Detecting unknown viruses
  - Undecidable -- Fred Cohen
  - Detect unknown viruses with acceptable accuracy

Conventional Virus Detection

- Signature matching
  - Simple signature
    - Long sequence makes the match inefficient
    - Short sequence results in a high false positive
  - Wild card Signature
    - Able to detect a family of virus
- Signature generation
  - Manual identification
    - Virus expert will read virus code and identify the unique string for the virus
  - Automatic extraction
    - Use a large database of benign software to filter out unique binary sequence of a virus

Common problem:
Can only detect known virus
Detecting Unknown Viruses

- The problem of detecting unknown virus is generally undecidable – *Fred Cohen*
- Detecting viruses with an acceptable detecting rate is possible
  - Static approach
    - Simple heuristic detection
    - N-Gram
    - Control flow analysis
  - Dynamic approach
    - Runtime system call trace
    - Dynamic instruction sequences

Data Mining Approaches in Virus Detection

|-------------------|-------------------|
| Feature | DM Model
| Binary | Neural Network |
| Trigram | |

<table>
<thead>
<tr>
<th>InSeon et al, 2004</th>
<th>Cai et al, 2005</th>
</tr>
</thead>
</table>
| Feature | DM Model
| Binary | SOM |
| Sequence | |

| Feature | DM Model
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>String Information</td>
<td>RIPPER</td>
</tr>
<tr>
<td>Binary Sequence</td>
<td>Naïve Bayes</td>
</tr>
<tr>
<td>Multi-Naïve Bayes</td>
<td></td>
</tr>
</tbody>
</table>

| Feature | DM Model
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
</tr>
<tr>
<td>Sequence</td>
</tr>
</tbody>
</table>
### Heuristics

- Entry point locates in the last section
- Suspicious section name
- Suspicious section characteristics
- Inconsistent size calculation
- Suspicious imports by ordinal
- Critical function imports
- Corrupt relocation table
- Occupation rate of last block in sections

### N-Gram

- N-Gram is consecutive bytes within executable file
  
  0400 B801 020E 07BB 0002 33
  C9 8BD1 419C

- 0400 B801 020E 07BB
- 00 B801 020E 07BB 00
- B801 020E 07BB 0002
- 01 020E 07BB 0002 33

- Abou-Assaleh et al, "Detection of New Malicious Code Using N-grams Signatures", PST’04

- Kolter, et al"Learning to detect malicious executables in the wild", SIGKDD 2004
**Static Instruction Sequences Approach**

**Assembly (Generated by IDAPro)**

```
loc_40102E:
  sub    eax, eax
  push   dword ptr fs:[eax]
  mov    fs:[eax], esp
  stc
  jb     short loc_401042
  jmp    edi, [esp+53Ch]
loc_401042:
  mov    edi, [esp+2Ch]
  jmp    short loc_401049
```

For every block in assembly, we generate a line of instructions

```
loc_40102E: sub push mov stc jb mov jmp
loc_401042: mov push mov stc
```

**Instruction Association**

- **Relationship of opcodes within a basic block**
- **Types of Instruction Association**

<table>
<thead>
<tr>
<th>Type</th>
<th>Continuous?</th>
<th>Ordered?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>2</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Instruction Sequences:

- Type 1: push sub
- Type 2: sub push
- Type 3: sub push

Instruction Association Sample
Feature Extraction

- Only frequent instruction association will be considered
  - Minimum support level: minimal frequency of instruction associations among all basic blocks
  - Use Apriori algorithm to extract frequent associations

```
push push push call test mov jnz
call mov push push mov jnz mov jmp
mov push push call mov test je push push push call test je
push push push push call push call
lea lea lea mov xor mov
call push push mov push push
call call mov cmp je inc
push push call mov
inc cmp jnz mov push push push push push push push push push call mov push push call pop push push call mov cmp je mov jnz pushad mov lea push or jmp
```

Feature Selection

- Selected features is discriminating
  - Frequent in benign code and rare in malicious code, or vice versa
- Contrast
  - We choose instruction associations with top contrast

\[
\text{Contrast}(F_i) = \begin{cases} 
\frac{\text{Count}_B(F_i) + \varepsilon}{\text{Count}_M(F_i) + \varepsilon} & \text{if}\ \text{Count}_B(F_i) \geq \text{Count}_M(F_i) \\
\frac{\text{Count}_M(F_i) + \varepsilon}{\text{Count}_B(F_i) + \varepsilon} & \text{if}\ \text{Count}_B(F_i) < \text{Count}_M(F_i) 
\end{cases}
\]

- \(\text{Count}_B(F_i)\): normalized count of \(F_i\) in benign instruction file
- \(\text{Count}_M(F_i)\): normalized count of \(F_i\) in malicious instruction file
- \(\varepsilon\): a small constant to avoid error when the dominant is 0
- \(F_i\): instruction association pattern
Dynamic Instruction Sequences Approach

OllyDbg Runtime Log (We capture first 40,000 lines)

```plaintext
0040538F Main MOV DWORD PTR DS:[EAX+8],EAX
00405392 Main MOV DWORD PTR DS:[EAX+4],EAX
00405395 Main ADD EAX,8 ; EAX=00A80914
00405398 Main DEC EDX ; EDX=00000038
00405399 Main JNZ SHORT AGRSMMSG.0040538F
```

Problem:
There are no generated blocks

Logic Assembly

```plaintext
Runtime log
0040538F Main MOV DWORD PTR DS:[EAX+8],EAX
00405392 Main MOV DWORD PTR DS:[EAX+4],EAX
00405395 Main ADD EAX,8
00405398 Main DEC EDX
00405399 Main JNZ SHORT AGRSM MSG.0040538F
```

Goal
- Remove duplicated instructions
- Generate basic blocks according to program semantics
- Result should be similar to assembly

Approach
- Sorting, remove duplication
- For every “jmp” destination, generate a label
Logic Assembly (Cont)

- Logic Assembly differ with Assembly in that
  - We can generate logic assembly for every executable; however, disassembling is not always successful
  - Logic assembly have less code coverage
  - Some labels are missing in logic assembly (According “jmp” instruction is not included in runtime log)

C4.5 Decision Tree

- Recursively split the dataset into parts
  - Use entropy gain as criteria
- Keep splitting until every node is only of one class
- Post pruning
  - Cut unnecessary split to avoid overfitting
  - Use validation dataset
Support Vector Machine

- Optimize a linear discriminant problem with error
- Map low dimensional space to high dimensional
  - Linear inseparable case maybe separable in high dimensional space
- libsvm, an open source C implementation of SVM

Dataset

<table>
<thead>
<tr>
<th>267 Win32 Viruses</th>
<th>368 benign executables</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Vx Heaven (vx.netlux.org)</td>
<td>Include</td>
</tr>
<tr>
<td>File virus</td>
<td>Windows system executables</td>
</tr>
<tr>
<td>Worm starter</td>
<td>Commercial executables</td>
</tr>
<tr>
<td>Trojan</td>
<td>Open source executables</td>
</tr>
<tr>
<td>Polymorphic and Metamorphic virus</td>
<td>Have the similar average size and variation as the malicious dataset</td>
</tr>
</tbody>
</table>

- We use 5 fold cross validation in our experiment
- Measurement
  - Accuracy
  - False Positive Rate
  - False Negative Rate
### Experimental Result

#### C4.5 decision Tree

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>FPR</th>
<th>FNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE consistency</td>
<td>90.0%/89.7%</td>
<td>2.6%/23.4%</td>
<td>2.8%/24.8%</td>
</tr>
<tr>
<td>8-Gram</td>
<td>98.6%/88.2%</td>
<td>0.9%/21%</td>
<td>11.1%/12.7%</td>
</tr>
<tr>
<td>Static IS</td>
<td>89.5%/86.3%</td>
<td>17.0%/20.6%</td>
<td>3.6%/6.6%</td>
</tr>
<tr>
<td>Dynamic IS</td>
<td>93.7%/91.0%</td>
<td>9.4%/12.6%</td>
<td>4.1%/6.5%</td>
</tr>
</tbody>
</table>

#### Support Vector Machine

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>FPR</th>
<th>FNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE consistency</td>
<td>90.6%/85.3%</td>
<td>2.2%/22.5%</td>
<td>1.7%/23.3%</td>
</tr>
<tr>
<td>8-Gram</td>
<td>94.5%/89.3%</td>
<td>2.0%/10.2%</td>
<td>6.0%/17.2%</td>
</tr>
<tr>
<td>Static IS</td>
<td>94.3%/84.9%</td>
<td>4.1%/12.7%</td>
<td>8.3%/18.7%</td>
</tr>
<tr>
<td>Dynamic IS</td>
<td>95.4%/91.9%</td>
<td>7.4%/9.6%</td>
<td>2.5%/6.8%</td>
</tr>
</tbody>
</table>

### Conclusion

- Data mining is effective in detecting unknown viruses
- Feature set plays a key role in data mining approach to detect unknown viruses
- Dynamic instruction sequences feature set achieve slightly better result than other feature sets we tested