Iterative System Call Patterns Blow the Malware Cover

Mansour Ahmadi, Ashkan Sami, Hossein Rahimi, Babak Yadegari
MansourAhmadi@gmail.com, asami@ieee.org
Shiraz University, Iran

“IT Security for the Next Generation”
Asia Pacific & MEA Cup 2011
University of Technology MARA, Malaysia
March 4-6, 2011
Outline

• Introduction
• Related Works
  ▶ Static & Dynamic Analysis
• Motivation
• Our Method
• Experiments
• Conclusion
• References
Introduction

• Anti-Viruses use signature-based method for detecting malware
• Polymorphic and metamorphic malware mutate their structures so anti-viruses regularly need to be updated
• We focused on MS windows portable executable (PE) malware
• Because of obfuscation, we should analyze PEs interactions with OS
• Our approach is based on data mining approach to facilitate and expedite the malware diagnosis
Related Works

- Malware Detection Based On Mining API Calls
  Ashkan Sami, Hossein Rahimi, Babak Yadegari, Naser Peiravian, Sattar Hashemi, Ali Hamze, (Shiraz University)

- Static, Mining API Calls from Header, Accuracy = 98%
- Obfuscation, Injecting Fake API Calls
Related Works (...)

- Effective and Efficient Malware Detection at the End Host
  Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou (Secure System Labs, California)

- Dynamic, Create Graph, Accuracy = 63%, high complexity

```c
GetModuleFileNameA([out] lpFilename -> "C:\netsky.exe")
...  
NtCreateFile(Attr->ObjectName:"C:\netsky.sks", mode: open, [out] FileHandle -> A)
...  
NtCreateFile(Attr->ObjectName:"C:\WINDOWS\AVprotect9x.exe", mode: create, [out] FileHandle -> B)
...  
NtCreateSection(FileHandle: A, [out] SectionHandle -> C)
NtMapViewOfFileSection(SectionHandle: C, BaseAddress: 0x3b0000)
...  
NtWriteFile(FileHandle: B, Buffer: "M2\90\00... ", Length: 1686)
...  
```
Related Works (…)

- Efficient Virus Detection Using Dynamic Instruction Sequences
  Jianyong Dai, Ratan Guha, Joohan Lee (University of Central Florida)

- Dynamic, Accuracy =91%

- Metamorphic Malwares replacing assembly instructions

![Diagram of Instruction Sequences and Assembly Instructions]

1. Original log
2. Logic Assembly
3. Abstract Assembly
4. Feature Selection
5. Classification Model
6. Decision
Motivation

• Detecting polymorphic and metamorphic malware in case of obfuscation.
• Prior works focused on graph isomorphism which is computationally very expensive.
• We wanted to generate graphs to detect recurring patterns in malware runs.
• Graph data mining is also very computationally expensive
• So we mined iterative patterns in sequences
• Replacing API Calls to evade detection is not easily possible.
Our Method

Controlled Environment

Malwares

Benigns

NewTrain Dataset

New Test Dataset

Classification

Prediction

Detection Engine

Select Discriminative Patterns

Mine Closed Patterns

Train Dataset

Test Dataset

APIs Log

<XML>

Windows DLLs

Predicted Samples
Monitoring

Backdoor.Win32.Agent.cy

RegOpenCurrentUser, OpenProcessToken, AllocateAndInitializeSid, CheckTokenMembership, FreeSid, RegOpenKeyExW, RegQueryValueExW, RegCloseKey, RegCloseKey, RegOpenKeyExW, RegOpenKeyExA, mmRegQueryValueExA, RegCloseKey, RegOpenKeyExW, RegOpenKeyExW, InitializeSecurityDescriptor, InitializeAcl, AddAccessAllowedAce, AddAccessAllowedAce, SetSecurityDescriptorDacl, MD4Init, MD4Update, MD4Update, MD4Update, MD4Final, OpenSCManagerA, OpenServiceA, CreateServiceA, StartServiceA, CloseServiceHandle, CloseServiceHandle
Pattern Matching

- Consider a pattern $P(<A, B>)$
- Database consisting of two PEs

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Sequence of API calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign1</td>
<td>$&lt;D, B, A, F, B, A, F, B, C, E&gt;$</td>
</tr>
<tr>
<td>Malware1</td>
<td>$&lt;D, B, A, D, B, B, B, A, B&gt;$</td>
</tr>
</tbody>
</table>

- $\text{Inst}(P)$ denote as the set of instances of $P$:
  $\{(1, 3, 5), (1, 6, 8), (2, 3, 5), (2, 8, 9)\}$
- Multiple occurrences of an iterative pattern are considered to reflect repetition of a behavior (e.g. all the worms copy themselves)
Closed Frequent Pattern

• An iterative pattern $P$ is **frequent** if its instances occur above a certain threshold of $\text{min\_sup}$ in $\text{APIDB}$, i.e.,

\[ \left| \text{Inst}(P, \text{APIDB}) \right| \geq \text{min\_sup} \]

• A frequent iterative pattern $P$ is **closed** if there exists no super sequence $Q$ s.t.:
  
  ▶ $P$ and $Q$ have the same support & $\text{Inst}(P) \approx \text{Inst}(Q)$

• To evaluate the discriminative power of a feature, statistical measure of **Fisher score** is adopted.
# Create Dataset

<table>
<thead>
<tr>
<th>Name Of PE</th>
<th>Single API1</th>
<th>Single API2</th>
<th>…</th>
<th>Closed Frequent API Pattern 1</th>
<th>Closed Frequent API Pattern 2</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign 1</td>
<td>Total of API1</td>
<td>Total of API2</td>
<td>…</td>
<td>Support</td>
<td>Support</td>
<td>…</td>
</tr>
<tr>
<td>Benign 2</td>
<td>Total of API1</td>
<td>Total of API2</td>
<td>…</td>
<td>Support</td>
<td>Support</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>Malware 1</td>
<td>Total of API1</td>
<td>Total of API2</td>
<td>…</td>
<td>Support</td>
<td>Support</td>
<td>…</td>
</tr>
<tr>
<td>Malware 2</td>
<td>Total of API1</td>
<td>Total of API2</td>
<td>…</td>
<td>Support</td>
<td>Support</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
### Experiments

- Random Forest with 10 Fold Cross Validation
- Results on 269 malware and 211 benign

<table>
<thead>
<tr>
<th>S</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>DR</th>
<th>ACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>243</td>
<td>26</td>
<td>179</td>
<td>32</td>
<td>90.33</td>
<td>87.91</td>
</tr>
<tr>
<td>0.15</td>
<td>247</td>
<td>22</td>
<td>180</td>
<td>31</td>
<td>91.8</td>
<td>88.95</td>
</tr>
<tr>
<td>0.2</td>
<td>247</td>
<td>22</td>
<td>180</td>
<td>31</td>
<td>91.8</td>
<td>88.95</td>
</tr>
<tr>
<td>0.25</td>
<td>245</td>
<td>24</td>
<td>173</td>
<td>38</td>
<td>91.1</td>
<td>87.08</td>
</tr>
<tr>
<td>0.3</td>
<td>247</td>
<td>22</td>
<td>171</td>
<td>40</td>
<td>91.8</td>
<td>87.02</td>
</tr>
<tr>
<td>0.35</td>
<td>242</td>
<td>27</td>
<td>177</td>
<td>34</td>
<td>90</td>
<td>87.29</td>
</tr>
<tr>
<td>0.4</td>
<td>243</td>
<td>26</td>
<td>176</td>
<td>35</td>
<td>90.3</td>
<td>87.29</td>
</tr>
</tbody>
</table>
False alarm rate

![Graph showing false alarm rate vs. minimum support](image)
Conclusion & future works

• Due to rapid growth and increasing malware, need for a mechanized system for detecting malware are bound.

• Anti-virus signature-based methods are reliable but not enough.

• Adding detection methods based on artificial intelligence can improve performance of anti-virus.

• We presented a method that find the best patterns for distinguishing malware and benign.

• We want to monitor more PEs with more DLLs and use complex structures for improving the results.
References


Thank You

Iterative System Call Patterns Blow the Malware Cover

Mansour Ahmadi, Ashkan Sami, Hossein Rahimi, Babak Yadegari,
MansourAhmadi@gmail.com, asami@ieee.org
Shiraz University, Iran

“IT Security for the Next Generation”
Asia Pacific & MEA Cup 2011
University of Technology MARA, Malaysia
March 4-6, 2011