Inoculation Against Malware Infection Using Kernel-level Software Sensors

Raymond Canzanese¹, Moshe Kam¹, and Spiros Mancoridis²

¹Dept. of Electrical and Computer Engineering
²Dept. of Computer Science
Drexel University
Philadelphia, PA, U.S.A.
Develop a method for accurate malware detection on a live system using classical detection theory
Overview

• Construct sensors to monitor software-OS interaction
• Choose a statistical model for the sensor data
• Build a detector to infer whether a system is infected
• Analyze detector performance using data sets from sensors on a live system
• Use canonical samples of malware to test detector on related malware
Motivation

- Signature-based detection is reactive
  - Malware must first be discovered and analyzed
  - Vulnerable to:
    - Zero-day malware
    - Obfuscated malware
    - Polymorphic malware
  - Example of the shortcomings: Zeus botnet
    - First found in wild in 2007
    - Used in successful Christmas 2010 attack
    - Illustrative of arms race involved in signature detection.
Related Work

• Automatic signature generation
  – Kephart, 1994
• API calls as features, Bayesian detection
  – Schultz, 2001
• n-gram analysis and machine learning
  – Kolter and Maloof, 2006
• Performance monitors as features, machine learning
  – Moskovitch, 2008
• Virtual machine sensors as features, computational geometry
  – Stehle, 2010
• API calls in a sandbox as features, clustering
  – Rieck, 2011
Contributions

• Malware detection using classical detection theory
• Computationally simple detector
  – Ease of training
  – Low overhead
• Sensors designed to monitor live systems
Sensor Overview

• Monitor 282 Windows XP OS functions
• Monitor all processes
• Log calls/second to each function
• Implemented as a Windows device driver
Detection system architecture
Statistical modeling

Data are Geometrically Distributed:

PMF: \( P(x) = (1 - p)^x p \)
Neymann-Pearson Detector

- Computes a likelihood ratio and compares it to a threshold
- Does not assign costs or assume \textit{a priori} probabilities
- Assumes the sensors and samples are independent
- Maximizes \( P_d \) given \( P_f \)

\[
\eta \geq \sum_{i=1}^{282} x_i \left( \log(1 - p_{c,i}) - \log(1 - p_{i,i}) \right)
\]
Testing scenarios

• **Webserver test**
  – **Software**: Apache and PHP
  – **2 sites**: Silverstripe and Drupal
  – **Load**: Replay actual access logs and Drupal test suite

• **Desktop computing test**
  – **Software**: OpenOffice.Org
  – **Load**: OpenOffice.Org test suite
Malware Samples

• Worms
  – Autoit
  – YahLover

• Trojans
  – Bybz
  – Bohu
  – Carberp
  – SpyEye

• Botnets
  – Darkness

Source: Offensive Computing @ http://offensivecomputing.net/
Data Sets

• 16 hours of clean data
• 56 hours of infected data
  – 8 hours for each malware sample

• Testing set:
  • 8 hours clean
  • 28 hours infected

• Training set:
  • 8 hours clean
  • 28 hours infected

Repeated for both the desktop and webserver loads
Single sensor difference in distribution

NtQuerySystemInformation

- **infected**
- **clean**

**density** vs. **calls/second**

6/16/2011
Single sensor detection performance

Empirical ROC Curve

\[ P_{\text{false alarm}} \]

\[ P_{\text{detection}} \]
Best
• CreateFile
• CreateKey
• QuerySystemInformation
• QueryValueKey
• SetInformationProcess

Worst
• Continue
• CreateSemaphore
• OpenThread
• QueryVirtualMemory
• Yield Execution
Webserver detector performance

Empirical ROC Curve

Area under curves:
- 0.9823
- 0.9937
- 0.9946
Detection rates for each test set against each training set:

<table>
<thead>
<tr>
<th>Training Set</th>
<th>Worms</th>
<th>Trojan Horses</th>
<th>Botnet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Autolt</td>
<td>Yahlover</td>
<td>Bohu.A</td>
</tr>
<tr>
<td>Autolt</td>
<td>--</td>
<td>12.8</td>
<td>93.4</td>
</tr>
<tr>
<td>Yahlover</td>
<td>99.9</td>
<td>--</td>
<td>99.8</td>
</tr>
<tr>
<td>Bohu.A</td>
<td>100.0</td>
<td>100.0</td>
<td>--</td>
</tr>
<tr>
<td>Bybz</td>
<td>97.9</td>
<td>90.0</td>
<td>68.4</td>
</tr>
<tr>
<td>Carberp</td>
<td>100.0</td>
<td>99.9</td>
<td>99.8</td>
</tr>
<tr>
<td>SpyEye</td>
<td>100.0</td>
<td>86.5</td>
<td>99.7</td>
</tr>
<tr>
<td>Darkness</td>
<td>100.0</td>
<td>99.9</td>
<td>99.8</td>
</tr>
</tbody>
</table>
Desktop computing detector performance
Hybrid detector performance

Empirical ROC Curve

- Blue dashed line: 5 samples
- Green dotted line: 10 samples
- Red solid line: 30 samples

P_detection vs. P_false alarm

6/16/2011
Conclusions and Future Work

• Neyman-Pearson test is effective for detecting malware
  – Computationally simple
  – Fast training and detection

• Effective inoculation achievable if training samples chosen correctly

• Future work:
  – Better statistical model for the data
  – Correlations among sensors
  – Expanded test cases and malware samples
  – Comparison with other detectors
Inoculation Against Malware Infection Using Kernel-level Software Sensors

Raymond Canzanese\textsuperscript{1}, Moshe Kam\textsuperscript{1}, and Spiros Mancoridis\textsuperscript{2}

\textsuperscript{1}Dept. of Electrical and Computer Engineering
\textsuperscript{2}Dept. of Computer Science
Drexel University
Philadelphia, PA, U.S.A.