Testing Malicious Code Detection Tools

Mihai Christodorescu
mihai@cs.wisc.edu

WiSA     http://www.cs.wisc.edu/wisa
University of Wisconsin, Madison
Problem

- **Wrong focus:** Testing looks only at today’s malware
  - What about tomorrow’s malware?

- **Efficacy:** How does one compare the efficacy of several malware detection tools?
  - Lack of openness about implementations
Our Solution

Test Case Generation through Obfuscation

• Generate new malware test cases
  – Using obfuscation transformations
  – Based on existing malware instances

• Test detection capabilities across a wide range of malware types
Milestones

✓ Binary rewriting infrastructure
  - For IA-32/Windows
  - For Visual Basic

⇒ Suite of obfuscations

⇒ Comparison metrics

Complete test suite
Overview

• Goals
• State of the art
• Our approach
• Testing environment
• Evaluation
• Conclusions
Testing Malware Detectors

Malware Detection Tool’s Goal:
Detect malicious code!

- **Focus:** *executable code that replicates*
  - Viruses, worms, trojans
  - Not buffer overflow attacks
  - Not spyware
- **Code is mobile**
Testing Goals

1. Measure detection rate for existing malware
   ➤ False negatives

2. Measure detection rate for benign software
   ➤ False positives

3. Measure detection rate of new malware
   ➤ Resilience to new malicious code
State of the art

• Several testing labs
  - Commercial
    • Virus Bulletin
    • International Computer Security Association (ICSA)
    • West Coast Lab’s CheckMark
  - Independent
    • University of Hamburg Virus Test Center (VTC)
    • University of Magdeburg
Sample certification req’s

Virus Bulletin 100% Award

• Detect all *In The Wild* viruses during on-demand and on-access tests
• *Generate no false positives*

ITW virus lists are maintained by WildList Organization International
Testing Goals

1. Measure detection rate for existing malware
   - False negatives
2. Measure detection rate of benign software
   - False positives
3. Measure detection rate of new malware
   - Resilience to new malicious code

✓ Checked
✓ Checked
???
Testing against Future Malware

• First attempt:
  Andreas Marx “Retrospective Testing”
  - Test 3- and 6-month old virus scanners

<table>
<thead>
<tr>
<th>Malware Type</th>
<th>Detection Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro viruses</td>
<td>74% - 94%</td>
</tr>
<tr>
<td>Script viruses</td>
<td>35% - 82%</td>
</tr>
<tr>
<td>Win32 file viruses</td>
<td>24% - 79%</td>
</tr>
<tr>
<td>Win32 worms, trojans, backdoors</td>
<td>8% - 37%</td>
</tr>
</tbody>
</table>
Testing against Future Malware

• We can learn from the past:
  – Often, old malicious code is slightly changed and re-launched

• Sobig e-mail worm

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fake “From”</td>
<td><a href="mailto:bill@microsoft.com">bill@microsoft.com</a></td>
<td><a href="mailto:admin@support.com">admin@support.com</a></td>
<td><a href="mailto:support@yahoo.com">support@yahoo.com</a></td>
</tr>
<tr>
<td>Distribution</td>
<td>Mass e-mail, Copy to shared drive</td>
<td>Mass e-mail, Copy to shared drive</td>
<td>Mass e-mail, Copy to shared drive</td>
</tr>
<tr>
<td>Deactivation</td>
<td>June 8</td>
<td>July 2</td>
<td>July 14</td>
</tr>
<tr>
<td>Update path</td>
<td>Geocities-hosted page</td>
<td>Hard-coded IPs</td>
<td>Hard-coded IPs</td>
</tr>
</tbody>
</table>
Overview

• Goals
• State of the art
• Our approach
• Testing environment
• Evaluation
• Conclusions
Obfuscation = Test Case Generation

• Obfuscate current known malware to obtain new malware test cases
• Why?
  - Many viruses / worms / trojans reuse code from older malware
  - Simulate self-mutating malware
  - Measure the ability of anti-virus tools to detect malicious behavior, not just malicious code instances
Test Case Generation

Collection of current malware

\[
\begin{pmatrix}
V_1 \\
V_2 \\
\vdots \\
V_n
\end{pmatrix}
\]

Set of obfuscation transformations

\[
\begin{pmatrix}
\sigma_1 & \sigma_2 & \ldots & \sigma_m
\end{pmatrix}
\]

New malware for testing

\[
\begin{pmatrix}
V'_{1,1} & \ldots & V'_{1,m} \\
V'_{2,1} & \ldots & V'_{2,m} \\
\vdots & & \vdots \\
V'_{n,1} & \ldots & V'_{n,m}
\end{pmatrix}
\]

- Garbage Insertion
- Code Reordering
- Interpreter
Test Case Generation

Collection of current malware

\[
\begin{bmatrix}
  V_1 \\
  V_2 \\
  \vdots \\
  V_n
\end{bmatrix}
\]

Set of obfuscation transformations

\[
\begin{bmatrix}
  \sigma_1 \\
  \sigma_2 \\
  \vdots \\
  \sigma_m
\end{bmatrix}
\]

New malware for testing

\[
\begin{bmatrix}
  V_{1,1} & \cdots & V_{1,m} \\
  V_{2,1} & \cdots & V_{2,m} \\
  \vdots & \ddots & \vdots \\
  V_{n,1} & \cdots & V_{n,m}
\end{bmatrix}
\]

Diagram:
- Binary
  - IDA Pro
    - Parse Binary
    - Build CFGs
- Library of obfuscations
- Connector
  - Memory Analysis
- BREW
  - Rewrite
- Generate Code
- Generated Binary
Sample Obfuscations

• Change data:
  – New strings, new dates, new constants
  – Encode / encrypt constants

• Change control:
  – Insert garbage
  – Encode / encrypt code fragments
  – Reorder code
  – Add new features
  – …
Sample Obfuscations

- Encode / encrypt constants

Message="Read this.."
Message.Send

Message=Decode("13FQ...")
Message.Send
...
Sub Decode(...)
...
## Earlier Obfuscation Results

*Commercial anti-virus tools vs. morphed versions of known viruses*

<table>
<thead>
<tr>
<th></th>
<th>Norton AntiVirus</th>
<th>COMMAND</th>
<th>McAfee VirusScan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chernobyl-1.4</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
</tr>
<tr>
<td>fOsfoR0</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
</tr>
<tr>
<td>Hare</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
</tr>
<tr>
<td>z0mbie-6.b</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
<td>☒ Not detected</td>
</tr>
</tbody>
</table>
Ideal Testing Results

Virus

\[ \sigma_1(A) \quad \sigma_2(A) \quad \sigma_3(A) \quad \sigma_4(A) \quad \sigma_5(A) \quad \sigma_6(A) \quad \sigma_7(A) \quad \sigma_8(A) \]

McAfee AV
Command AV
Norton AV
Parametrized Obfuscation

Encoding data:

Message = “Read this..”
Message . Send

Parameters:

• Data to obfuscate
• Type of encoding / encryption
• Encryption key
• Location of obfuscation

Message = Decode (“13FQ...”)
Message . Send
...
Sub Decode (...)
Testing Environment

• Simple and complex obfuscations applied to known (detected) malware

• **Multiple malware detection tools:**
  - McAfee
  - Norton
  - Sophos
  - Kasperski
Overview

• Goals
• State of the art
• Our approach
• Testing environment
• Evaluation
• Conclusions
Evaluation

• Test against a set of malware detected in original form by all tools
• Test using the same obfuscations and the same parameters

• Obfuscation hierarchy
  - From simple to most complex
Metrics

• **Minimum obfuscation level:**
  For a given obfuscation $\sigma$ with parameters $(x_1, ..., x_k)$, what are the least values for each $x_i$ that generates a false negative?

• **Minimal combination of obfuscations:**
  What is the smallest set of obfuscations $\{\sigma_1, ..., \sigma_k\}$ that generates a false negative?
Preliminary Results

AnnaKournikova
Encode 1
Encode 5
Encode 9
Encode 13
Encode 17
Encode 21
Encode 25
Encode 29
McAfee AV
Command AV
Norton AV
Lessons Learned

• Malware spreads instantaneously
  – Arms race between malware distribution and malware detection tool update

• In testing malware detection tools, one must use a virus writer’s mindset
  – It is a game → need to think several moves ahead
Future Work

• Explore more obfuscations
  Depends on results of ongoing tests
• Future idea # 1:
  Self-guided test tool that finds the minimal test cases for false negatives
• Future idea # 2:
  Develop tests to check for detection of malicious behavior, not just code sequences
Seeing Through the Obfuscations

Smart Virus Scanner

Pattern Library

Annotator

Annotated Program

Malicious Code Blueprint

Detector

Yes/No
Detection Example

Virus Code:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>(from Chernobyl CIH 1.4 virus)</th>
</tr>
</thead>
<tbody>
<tr>
<td>push    eax</td>
<td></td>
</tr>
<tr>
<td>sidt    [esp-02h]</td>
<td></td>
</tr>
<tr>
<td>pop     ebx</td>
<td></td>
</tr>
<tr>
<td>add     ebx, HookNo * 08h + 04h</td>
<td></td>
</tr>
<tr>
<td>cli     ebp, [ebx]</td>
<td></td>
</tr>
<tr>
<td>mov     bp, [ebx-04h]</td>
<td></td>
</tr>
<tr>
<td>lea     esi, MyHook - @1[ecx]</td>
<td></td>
</tr>
<tr>
<td>push    esi</td>
<td></td>
</tr>
<tr>
<td>mov     [ebx-04h], si</td>
<td></td>
</tr>
<tr>
<td>shr     esi, 16</td>
<td></td>
</tr>
<tr>
<td>mov     [ebx+02h], si</td>
<td></td>
</tr>
<tr>
<td>pop     esi</td>
<td></td>
</tr>
</tbody>
</table>

Virus Automaton:

Irrelevant instruction

Virus Found!
References

Mihai Christodorescu, Somesh Jha “Static analysis of executables to detect malicious patterns”. USENIX Security’03, August 2003, Washington DC.

Andreas Marx “Retrospective testing - how good heuristics really work”. Virus Bulletin Conference, November 2002, New Orleans, LA.

Binary Code Rewriting

Binary

IDA Pro
- Parse Binary
- Build CFGs

Connector
- Memory Analysis
- BREW
- Rewrite

Codesurfer
- Build SDG
- Browse

Clients
- Detect Malicious Code
- Detect Buffer Overrun
- Build Program Specification

Generated Binary