PROVENANCE-BASED INTEGRITY PROTECTION FOR WINDOWS

Wai Kit Sze and R. Sekar
Stony Brook University

This work was supported in part by NSF (grants CNS-0831298 and CNS-1319137) and DARPA (contract FA8650-15-C-7561).
Total number of unique malware increases exponentially
Existing defenses

• Antivirus
  • Not effective against novel malware
• Behavior monitoring
  • Malware can often “fly under the radar”
• Sandboxing/protected view/broker architecture
  • Sandbox escape attacks

• Result: Sophisticated malware can chain exploits to escape all deployed defenses

Demo
• Chaining Adobe Reader exploit + Stuxnet to bypass existing defenses
Adobe + Stuxnet

Unknown Sources

Antivirus

Sandboxing

Arbitrary Code Execution

Behavior Monitoring

abc.Ink
Our Approach: Provenance Tracking

- Key idea for breaking such attacks
  - Track malware effects, prevent benign code from being tricked by them
How to realize?

• Design brand new OS (HiStar, Asbestos)?
  • New abstractions for capturing provenance and fully customizable policy enforcement
  • No adoption and hard to evaluate technique effectiveness with real malware

• Build mechanisms in today’s OSes (Flume, PPI, UMIP, IFEDAC)?
  • Work with open-source OSes
  • Questionable completeness: Are all resources tracked? Are all hooks implemented properly?

• Reuse existing mechanisms for both tracking and policy enforcement?
  • Portable, support contemporary OSes where source code is not available
  • Lack of flexibility => poor usability?
SPIF

Avoid changes

• Same **OSes**
  • Implemented on **Windows XP, 7, 8, and 10**

• Same **COTS applications**
  • Support Microsoft Office, Adobe Reader, Google Chrome, Photoshop CC, …

• Same **policy** for all applications
  • Simple, effortless policy development

• Same user **experience**
  • Minimize changes to user interactions
Tracking Provenance with userid

- Every resource in OSes is already labeled using userid

```
Benign User
  ↓
Untrusted User
  ↓
Benign User
```

```
Benign User
  ↓
Untrusted User
```

System

8
Dual-sandbox architecture

**Untrusted Sandbox**
Confine untrusted processes from attacking benign processes *directly*
- No signaling/IPC with benign processes
- Cannot modify benign files

**Benign Sandbox**
Confine benign processes from consuming untrusted resources *accidentally*
- No IPC with untrusted processes
- Read/access no untrusted files

- Create files in user directory
- Open user files for reading
- ...

✗ ✗ ✔
Properties of Dual-sandbox architecture

• Simple generic policy
  • Each sandbox blocks impermissible flows from its *own perspective*: No guessing

• Usable yet secure policy
  • Policies are *enforced at two stages* (avoid being overly conservative)
    • Permit untrusted operations that do not directly compromise benign processes
    • Example: Create new files in user directories

• Preserve *normal user experience*
  • Data from benign and untrusted can be saved in user directories, just as they are on today’s OSes
Dual-sandbox architecture

Processes can be malicious and be actively seeking ways to compromise benign processes

- Run untrusted processes as a different user
- Non-bypassable & robust policy enforcement mechanism

Processes have no intention to circumvent the protection mechanism

- Wrap calls to protect benign processes
- Circumventable, but no reason or incentive for benign processes to do so
Implementation

• 4000 lines of C++, 1500 lines of header
• Initially developed on Windows XP, then on Windows 8.1. Also tested on Windows 7 specifically for the high profile Sandworm malware
• Handled both files and registry entries
• Instead of modifying libraries, relies on Detours to hook on Windows APIs dynamically
  • Bootstrap using App_Init (Requires loading of User32.dll)
  • Propagate via library injection

Compatible with all of the applications tested!
Micro-benchmarks

- Postmark (File I/O):

<table>
<thead>
<tr>
<th>File Size</th>
<th>Benign</th>
<th>Untrusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small (500B-5KB)</td>
<td>6.53%</td>
<td>12.38%</td>
</tr>
<tr>
<td>Medium (5KB-300KB)</td>
<td>3.05%</td>
<td>4.58%</td>
</tr>
<tr>
<td>Large (300KB-3MB)</td>
<td>1.27%</td>
<td>-0.62%</td>
</tr>
</tbody>
</table>
Micro-benchmarks

- **SPEC2006 (CPU):**

<table>
<thead>
<tr>
<th></th>
<th>Unprotected Time (s)</th>
<th>Benign Overhead</th>
<th>Untrusted Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>401.bzip2</td>
<td>1785.9</td>
<td>-0.33%</td>
<td>0.26%</td>
</tr>
<tr>
<td>429.mcf</td>
<td>716.4</td>
<td>-1.69%</td>
<td>-0.96%</td>
</tr>
<tr>
<td>433.milc</td>
<td>3314.1</td>
<td>1.15%</td>
<td>-0.53%</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>1094.9</td>
<td>0.26%</td>
<td>-0.08%</td>
</tr>
<tr>
<td>450.soplex</td>
<td>1108.0</td>
<td>0.58%</td>
<td>2.34%</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>2386.2</td>
<td>0.02%</td>
<td>0.13%</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>1442.5</td>
<td>-0.25%</td>
<td>0.20%</td>
</tr>
<tr>
<td>470.lbm</td>
<td>1203.0</td>
<td>-1.51%</td>
<td>-0.32%</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>750.9</td>
<td>0.96%</td>
<td>1.83%</td>
</tr>
<tr>
<td>482.sphinx3</td>
<td>2653.6</td>
<td>-2.55%</td>
<td>-3.45%</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td></td>
<td><strong>-0.336%</strong></td>
<td><strong>-0.059%</strong></td>
</tr>
</tbody>
</table>
Firefox Loading Top 1000 Alexa Pages

Benign Overhead: 3.32%

Untrusted Overhead: 3.62%
Protection against malware

- Samples from Metasploit published in 2014 that can be exploited in our testbed

<table>
<thead>
<tr>
<th>CVE/OSVDB-ID</th>
<th>Application</th>
<th>Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-4696</td>
<td>WinAmp</td>
<td>Preference file (ini)</td>
</tr>
<tr>
<td>2013-3934</td>
<td>Kingsoft Office</td>
<td>Data (wps)</td>
</tr>
<tr>
<td>2010-2568</td>
<td>Windows Explorer</td>
<td>Data (lnk) &lt;- Stuxnet</td>
</tr>
<tr>
<td>104141</td>
<td>Calavera Uploader</td>
<td>Preference file (dat)</td>
</tr>
<tr>
<td>2014-2013</td>
<td>MuPDF</td>
<td>Data (xps)</td>
</tr>
<tr>
<td>102205</td>
<td>CCProxy</td>
<td>Preference file (ini)</td>
</tr>
<tr>
<td>100619</td>
<td>Total Video Player</td>
<td>Preference file (ini)</td>
</tr>
<tr>
<td>2013-6874</td>
<td>Light Alloy</td>
<td>Data (m4u)</td>
</tr>
<tr>
<td>2014-0568</td>
<td>Adobe Reader</td>
<td>Code</td>
</tr>
<tr>
<td>2014-4114</td>
<td>Microsoft Windows</td>
<td>Data (ppsx) &lt;- Sandworm</td>
</tr>
</tbody>
</table>
Conclusion

• We proposed SPIF, an integrity protection system to protect against malware on Windows.
• SPIF requires *no modification* to OS, applications, and user experience.
• SPIF defends against *high-profile malware* such as Stuxnet and Sandworm.

• Installation package & source code are available for downloading at http://seclab.cs.sunysb.edu/seclab/download.html. Comment and feedback welcome!
QUESTIONS
THANK YOU