AUTOVAC: Towards Automatically Extracting System Resource Constraints and Generating Vaccines for Malware Immunization

An ICDCS ’13 paper

Zhaoyan Xu   Jialong Zhang   Guofei Gu   Zhiqiang Lin

Texas A&M University

19 July 2013
Overview

- Prevent malware infection by exploiting malware’s own “de-duplication.”
  - Automatically extract the system resource constraints from malware code...
  - ...and generate vaccines based on the system resource conditions.
- Approach.
  - Monitor data propagation from system-resource-related system calls.
  - Automatically identifies the environment related state of a computer.
  - Analyzing the environment state, automatically generate vaccines.
- Evaluate on a large malware corpus and successfully extracted working vaccine for, e.g., Conficker, Sality and Zeus.
Procedure

- **Candidate identification.**
  - Trace, dynamic taint analysis, and obtain candidate resources that affect control flow.
  - Malware samples that are resource-insensitive are filtered.

- **Vaccine generation.**
  - Exclusiveness analysis: Exclude resources that are not unique to malware to prevent false positives.
  - Impact analysis: Program alignment to compare execution differences (impact) under different resource condition.
  - Determinism analysis: Backward taint analysis and program slicing to understand identifier generation logic.

- **Vaccine deployment:** Either **direct injection** or using **vaccine daemon**.
Candidate identification
Dynamic taint analysis.

- **Taint source**: Origin of tainted data.
  - **Criteria**: Unique presence (exclude transient events), minimal impact to system, easy deployment.
  - **Resources chosen**: mutex, files, registry entries, processes, library, GUI window and service.
  - **Instruction level**: Resource identifiers in Windows API, e.g., NtQueryObject.

- **800 Windows APIs**: Taint API function arguments and return values.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>OpenMutex</th>
<th>ReadFile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutex</td>
<td>3rd parameter: lpName</td>
<td>1st parameter: hFile for Handle Map</td>
</tr>
<tr>
<td>Success</td>
<td>EAX: Valid Handle Value</td>
<td>EAX: TRUE</td>
</tr>
<tr>
<td>Failure</td>
<td>EAX: NULL, GetLastError: 0x02</td>
<td>EAX: FALSE GetLastError: 0x1E</td>
</tr>
</tbody>
</table>

- **Taint propagation**.
  - For each instruction, taint destination if a source is labeled.
  - For comparison, a candidate resource for vaccine, pass to next phase.
Vaccine generation
Exclusiveness analysis.

- Exclude shared resources (e.g., mscrt.dll) with benign programs.
- Tell from the Google search result of resource ID.
Vaccine generation
Impact analysis.

- Run malware again in a controlled environment that can control return/arguments (one at a time), and compare normal execution.
- How compare traces: Program alignment problem (compare only unaligned instructions).

**Algorithm 1** Differential Analysis on the API-Call Traces

\( \Pi_m \): Manipulated Call Trace, \( \Pi_n \): Natural Call Trace  
\( \Delta_m \): Unaligned Call Trace in \( \Pi_m \), \( \Delta_n \): Unaligned Call Trace in \( \Pi_n \),  
\( f_\Pi \): (name, caller eip, parameter list), \( f_\Delta \): (name, parameter list)

```
1: \( \Delta_m \leftarrow \emptyset, \Delta_n \leftarrow \emptyset \)
2: for call \( f_{\Pi_m} \) in \( \Pi_m \) do
3:   for call \( f_{\Pi_n} \) in \( \Pi_m \) do
4:     if isAligned(\( f_{\Pi_m}, f_{\Pi_n} \)) then
5:       GO TO FIND_ALIGNED
6:   end if
7: end for
8: \( \Delta_m = \Delta_m \cup f_{\Delta_m} \)
9: end for
10: \( \Delta_n = \Pi_n \)
11: FIND_ALIGNED:
12: \( \Delta_n = \Pi_m[0, index(f_{\Pi_n})] \)
13: \( \{ f_{\Delta_i} \} = \text{Diff}(\Delta_m, \Delta_n) \)
14: return \( \{ f_{\Delta_i} \} \)
```

- Classify.
  - Full immunization: If the diff contains exit API, e.g., ExitThread, TerminateProcess.
  - Partial immunization: Impact can be told from diff, e.g., disable kernel injection, disable mass size network behavior, disable malware persistence, disable benign process injection.
  - No immunization.
Vaccine generation
Determinism analysis.

Determine whether the resource ID can be predicted and preempted acquired by vaccine.
Vaccine deployment

- Direct injection: for **fully** deterministic resource-sensitive malware.
- Vaccine daemon: for **algorithmically** deterministic resource-sensitive malware.
References.

A. Experiment Dataset

Our test dataset consists of 1,716 malware samples, which are collected from multiple online malware repositories (e.g., [1], [3], [4]) with mostly from Anubis [1]. We also leverage an online malware classification tool, VirusTotal [6], to obtain the classification information for these malware. We summarize classification results in Table II. We can see that these malware samples fall into 6 categories such as Backdoor (722 samples), Downloader (574 samples) and Trojan (184 samples).
Figure 3. Statistics on Malware’s Resource Sensitive Behaviors
## References.

<table>
<thead>
<tr>
<th>Vaccine Type</th>
<th>Backdoor</th>
<th>Trojan</th>
<th>Worm</th>
<th>Adware</th>
<th>Downloader</th>
<th>Virus</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>33%</td>
<td>27%</td>
<td>24%</td>
<td>30%</td>
<td>45%</td>
<td>81%</td>
</tr>
<tr>
<td>Registry</td>
<td>15%</td>
<td>29%</td>
<td>21%</td>
<td>13%</td>
<td>20%</td>
<td>19%</td>
</tr>
<tr>
<td>Windows</td>
<td>3%</td>
<td>14%</td>
<td>0%</td>
<td>47%</td>
<td>11%</td>
<td>0%</td>
</tr>
<tr>
<td>Mutex</td>
<td>8%</td>
<td>12%</td>
<td>29%</td>
<td>0%</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>Process</td>
<td>8%</td>
<td>7%</td>
<td>14%</td>
<td>0%</td>
<td>10%</td>
<td>0%</td>
</tr>
<tr>
<td>Library</td>
<td>26%</td>
<td>9%</td>
<td>4%</td>
<td>0%</td>
<td>7%</td>
<td>0%</td>
</tr>
<tr>
<td>Service</td>
<td>7%</td>
<td>2%</td>
<td>8%</td>
<td>10%</td>
<td>5%</td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deployment</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>67%</td>
<td>79%</td>
<td>63%</td>
<td>69%</td>
<td>69%</td>
<td>84%</td>
</tr>
<tr>
<td>Daemon</td>
<td>33%</td>
<td>21%</td>
<td>37%</td>
<td>31%</td>
<td>31%</td>
<td>16%</td>
</tr>
</tbody>
</table>

### Table V

**Vaccine Statistics on Different Malware Families**
Comments

» Try not make the same mistake as the malware: Try to be smart but not smart enough.
   » Smart: Check resources for de-duplication (so do not degenerate into a Morris worm).
   » Not smart enough: The resource can be predicted and captured by defenders.

» A lesson for MTD in virtmon design.
   » Making the interface a moving target.
   » Defense in the depth: Has a back up if MTD fails, i.e., resync mechanism.

» Example: Hypercall.
   » virtmon hypercall interface: service, arg1, arg2, token.
   » token change with each new hypercall, compute from the XOR sum of past, say, 4 hypercalls arguments.
   » Even if malware guess correctly once (which is very unlikely given the guessing space), it cannot guess more than that.
   » Limited-chance resync: If out-of-sync due to accidents or malware interference, can resync by using passing the last 2 successful hypercall tokens.