Towards Practical Application-level Support for Privilege Separation

Nik Sultana
Illinois Institute of Technology

Henry Zhu
UIUC

Ke Zhong
University of Pennsylvania

Zhilei Zheng
University of Pennsylvania

Ruijie Mao
University of Pennsylvania

Digvijaysinh Chauhan
University of Pennsylvania

Stephen Carrasquillo
University of Pennsylvania

Junyong Zhao
University of Pennsylvania

Lei Shi
University of Pennsylvania

Nikos Vasilakis
Brown University & MIT

Boon Thau Loo
University of Pennsylvania

ACSAC’22
Motivation: Software Security

Increased trend in # of CVEs:
Good: we know about problems.
Bad: there are more problems.

Ack: Graph generated using dataset from https://www.cve-search.org/dataset/
Software Security Techniques

• Range of techniques available: ASLR, Stack canaries, Sandboxing, Soft/hard bounds checking, ...

• Combining them is good practice. But some techniques are difficult to apply.

We focus on one such technique: privilege separation.
What is Privilege Separation? (privsep)

- Compartmentalize code + data. Early application: SSH server.
- Monolithic application: often common privileges throughout.
- Distributed system: granularity of privilege allocation.

<table>
<thead>
<tr>
<th>Application</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileges</td>
<td></td>
</tr>
</tbody>
</table>
What is Privilege Separation? (privsep)

- **Compartmentalize code + data.** Early application: servers: SMTP, SSH.
- **Monolithic application** → Concurrent set of cooperating programs.
  - **Monolithic application:** often common privileges throughout.
  - **Distributed system:** granularity of privilege allocation.

Heuristics for splitting software.
Why

What is Privilege Separation? (privsep)

• **Compartmentalize code + data.** Early application: servers: SMTP, SSH.

• **Monolithic application** → Concurrent set of cooperating programs.

Main benefit: **vulnerability containment.**
Best case: if a vulnerability is exploitable, then fewer privileges can be abused.
Implementing Privsep

- **Implementing** privsep: usually a lot of work. Restructuring logic and code, positive and negative tests.

- Changing software without introducing bugs!

- There are many **decisions** to take (and retake later) wrt what+how to separate.
Implementing privsep: usually a lot of work. Restructuring logic and code, positive and negative tests.

- Changing software without introducing bugs!

- There are many **decisions** to take (and retake later) wrt what+how to separate. *(See yellow bubbles above)*
What Privsep looks like

- Distributed system, heterogeneous privileges.

  Sometimes: separating between trusted vs untrusted.

Some parts are buggy?

*Fewer privileges = fewer problems.*
What Privsep looks like

Heuristics:
- Components needing specific access.
- Dependencies incl. libraries.
- Cross-domain interfaces (e.g., parts of network, filesystem)
Privsep, and then?

- **Drawbacks** include:
  Inertia wrt splitting software, introduction of new failure modes (hello distributed systems), performance overhead, inertia wrt maintainability and portability (e.g., if use hardware enforcement).
Research Goal

Widely-applicable tool support for privsep

Foundations:
- compartment model
- tool infrastructure
- software-level

(Longstanding)

(This paper)
Research Goal

Widely-applicable tool support for privsep

Foundations:
- compartment model
- tool infrastructure
- software-level

Artefacts:
+ tooling
+ several examples
+ supporting scripts & documentation

(This paper)
What’s different from prior art?

- **Separation “distance” + flexibility.**
  Separate binaries vs separate processes.
  Number of compartments.
  Commodity kernels and hardware.

- **Both tool and library.**
  Either can be used directly.
  Tool adapts code to use library.

- **Model-based approach.**
  Implemented abstractions provided/explained by the model.
The system has two components based on a model:

- Pitchfork 1 2
- libcompart 3
The **system** has two components based on a **model**:

- Pitchfork **1** **2**
- libcompart **3**

The **model** supports:

- Multiple compartments (different levels of trust)
- Synchronous communication
- Monitoring and failure-handling
if(console_type == BEEP_TYPE_CONSOLE) {
  pitchfork_start("Privileged");
  if(ioctl(console_fd, KIOCSOUND, period) < 0) {
    putchar(\'\a\'); /* Output the only beep we can, in an
                   effort to fall back on usefulness */
    perror("ioctl");
  }
  pitchfork_end("Privileged");
} else {
  /* BEEP_TYPE_EVDEV */
  struct input_event e;
  e.type = EV_SND;
  e.code = SND_TONE;
  e.value = freq;
  pitchfork_start("Privileged");
  if(write(console_fd, &e, sizeof(struct input_event)) < 0) {
    putchar(\'\a\'); /* See above */
    perror("write");
  }
  pitchfork_end("Privileged");
}
Compartment Model

- domain0
  - compart2
  - compart3
  - main

- domain1
  - compart1
  - monitor
  - tifftopnm
  - cmdparse
  - libtiff

- domain2
  - monitor
  - main

- temps
- topnm
Example of what’s enabled

- Machine and network-level policy enforcement.
- Communication channel over TCP.
- Organization:
  - **Domain**: one on each machine
  - **Compartments**: one in each domain.
  - **Segments**: 2 in Classified, 1 in Main.
Evaluation

(Many more details in the paper)

• Applicability
  • Examples
  • Maintainability
  • Convenience

• Security
  • Known CVEs
  • Heuristics

• Overhead: running time, memory, binary size.
Evaluation

- Applicability
- Examples
- Maintainability
- Convenience
- Security
- Known CVEs
- Heuristics
- Overhead: running time, memory, binary size.

<table>
<thead>
<tr>
<th>Software Plat.</th>
<th>Separation Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>cURL</td>
<td>L Command invocation, parsing, file transfer.</td>
</tr>
<tr>
<td>Evince</td>
<td>L libspectre dependency—see §2.</td>
</tr>
<tr>
<td>git</td>
<td>L Historical vulnerability [13].</td>
</tr>
<tr>
<td>ioquake3</td>
<td>m Applying server updates.</td>
</tr>
<tr>
<td>tifftopnm</td>
<td>L Separating parsers—see §C.</td>
</tr>
<tr>
<td>nginx</td>
<td>L HTTP request parsing</td>
</tr>
<tr>
<td>redis</td>
<td>L Isolating low-use commands.</td>
</tr>
<tr>
<td>tcpdump</td>
<td>F Leveraging Capsicum [68].</td>
</tr>
<tr>
<td>uniq</td>
<td>}</td>
</tr>
<tr>
<td>Vitetris</td>
<td>L Network-facing code—see §2.</td>
</tr>
</tbody>
</table>
Evaluation

- Applicability
- Examples
- Maintainability
- Convenience
- Security
- Known CVEs
- Heuristics

Overhead: running time, memory, binary size.

\[
SAR = \frac{\text{#LOC Synthesized}}{\text{#Lines of Annotation}}
\]

<table>
<thead>
<tr>
<th>Soft.</th>
<th>#LOC</th>
<th>#Annot.</th>
<th>#LOC Synthesized</th>
<th>SAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>beep</td>
<td>372</td>
<td>9</td>
<td>133</td>
<td>42</td>
</tr>
<tr>
<td>PuTTY</td>
<td>123K</td>
<td>6</td>
<td>52</td>
<td>13.5</td>
</tr>
<tr>
<td>wget\textsuperscript{6}</td>
<td>62.6K</td>
<td>3</td>
<td>65</td>
<td>77.7</td>
</tr>
<tr>
<td>wget\textsuperscript{7}</td>
<td>62.8K</td>
<td>8</td>
<td>57</td>
<td>11.9</td>
</tr>
</tbody>
</table>
Evaluation

- Applicability
- Examples
- Maintainability
- Convenience
- Security
- Known CVEs
- Heuristics

<table>
<thead>
<tr>
<th>Software</th>
<th>CVE-<em>-</em></th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>beep</td>
<td>2018-0492</td>
<td>Race condition</td>
</tr>
<tr>
<td>PuTTY</td>
<td>2016-2563</td>
<td>Stack buffer overflow</td>
</tr>
<tr>
<td>wget</td>
<td>2016-4971</td>
<td>Arbitrary file writing</td>
</tr>
<tr>
<td>wget</td>
<td>2017-13089</td>
<td>Stack buffer overflow</td>
</tr>
</tbody>
</table>

- Overhead: running time, memory, binary size.
System release

- [http://pitchfork.cs.iit.edu](http://pitchfork.cs.iit.edu)

- Everything is released except for exploit code:
  - libcompart
  - Pitchfork
  - examples of applying libcompart & Pitchfork
  - FreeBSD ports analysis
- Apache 2.0 license