Practical Whole-System Provenance Capture

Thomas Pasquier, Xueyuan Han, Mark Goldstein, Margo Seltzer (Harvard University), Thomas Moyer (MIT Lincoln Laboratory), David Eyers (University of Otago), Jean Bacon (University of Cambridge)
What is provenance?

- From the French “provenir” meaning “coming from”
- **Formal set of documents** describing the origin of an art piece
- **Sequence** of
  - Formal ownership
  - Custody
  - Places of storage
- Used for authentication
What is data-provenance?

- Represent interactions between objects of different types
  - Data-items (entities)
  - Processing (activities)
  - Individuals and Organisations (agents)
- Represented as a directed acyclic graph (think information flows)
- Edges represent interactions between objects as dependencies
- It is a representation of history
  - Immutable (unless it’s 1984)
  - No dependency to the future
Example
Why do we care about data-provenance?

- Originally in *reproducibility/repeatability/benefaction*
  - Can I reproduce those results?
  - Can I reproduce the computational environment?
- Application to *retrospective security*
  - Assuming highly trusted agents (think doctor)
  - Critical system (patient life vs access control error)
  - We may still need to understand when things went wrong
- Access control based on origin of data
  - *Provenance-based access control* (PBAC) Sandhu et al. PST’12
- And more…
  - Audit, *intrusion detection*, source of context for search algorithm, fault injection, *compliance auditing* etc…
Provenance in Operating Systems

- How is provenance captured?
  - **Application level**: capture application logic | fine grained
  - **System level**: complete and systematic | coarse grained
  - Both **PASS v2** (Seltzer et al. USENIX ATC 2009)

- How to implement in Linux?
  - Modify FS and SysCall interface? (**PASS** USENIX ATC 2006)
  - Develop a framework akin to LSM? (**LPM** USENIX Sec. 2015 Bates et al.)
  - Modify standard library that interface with sys-calls? (**OPUS**, USENIX TaPP. 2013 Balakrishnan et al.)

- Provenance is useful...
- ... but it is not being widely adopted
- Hard to deploy? No accessible out of the box solution?
Building a **practical** capture mechanism

---

- “Benefaction” problem (Collberg et al. Com. of ACM 2016)
- Maintainability issues, previous implementation hard to port to modern kernel
- Lack of clear separation of concerns between **capture**, **storage** and **query** (SPADE did great there)
- Objectives:
  - Clear focus on **capture**
  - Easy **integration** with other concerns
  - **Self-contained** mechanism
  - **Systematic and ubiquitous** capture
Use-case driven development

- Understanding how people want to use provenance
- We developed features based on **concrete use-cases**
- Based on the literature...
- ... or ongoing research projects
- Focus on **usability**
- We build a capture system...
- ... **clear separation of concern** vs holistic approach
Plan

- Maintainability
- Selective provenance capture
- Embedded capture

- More at the poster session...
Maintainability

- Leverage existing kernel features whenever possible
- Avoid alteration of existing code
- We therefore build upon:
  - **Linux Security Module**
    - to capture system events
  - **NetFilter**
    - to capture network events
  - **RelayFS**
    - to transfer provenance to user space
  - **SecurityFS**
    - to provide a userspace interface for settings
### Extent of modification

---

Existing files modified - total line of code for Prov Capture

<table>
<thead>
<tr>
<th>System</th>
<th>Headers</th>
<th>C File</th>
<th>Total</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS (v2.6.27)</td>
<td>18</td>
<td>69</td>
<td>87</td>
<td>5100</td>
</tr>
<tr>
<td>pub. 2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPM (v2.6.32)</td>
<td>13</td>
<td>61</td>
<td>74</td>
<td>2294</td>
</tr>
<tr>
<td>pub. 2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CamFlow (v4.9.5)</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td>2428</td>
</tr>
<tr>
<td>pub. 2017</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When porting CamFlow, only need to worry about internal API changes (more details in the paper).
Selective provenance

- System-provenance guarantee completeness...
- ... but generates too much data
- Previous address capture problem ...
  - capture everything we can
- ... but replace it with a storage and query problem
- Solution **capture only what you need**
Selective provenance

- Define capture target based on
  - Pathname (track any transformation to my dataset)
  - Process (track the action of this process)
  - Network interface (track interaction with server X)
  - Security ID (track interaction within this security domain)
  - and more...

- Propagate tracking to derivative data-item
- Only the relevant information is being captured
- Significantly reduce amount of data generated
Embedding queries in the capture mechanism

- **Data Loss Prevention** (Bates et al. USENIX security 2015)
  - Run only a well defined set of queries
  - Most information captured is irrelevant to the query...
  - ... but response time is at best $O(S)$ where $S$ is the total graph size
  - $S$ increase linearly overtime
- **Take only what you need?** (Bates et al. TaPP 2015)
  - $S$ smaller...
  - ... but still increase overtime
Embedding queries in the capture mechanism

- Provenance <-> causality graph
- **Propagate labels** along the graph at capture time
  - Query generally of the type find PATH along X, Y, Z
- Propagation follow query requirements
- Response time $O(1)$
  - Is label(s) present?
- Propagation incurs very small runtime overhead
- Future work **programmable query**...
- Complete provenance graph as **forensic** evidence
  - Response no more dependent on S!
- Framework in development!
Performance overhead

---

Micro-benchmark

<table>
<thead>
<tr>
<th>Sys Call</th>
<th>Whole</th>
<th>Selective</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat</td>
<td>100%</td>
<td>28%</td>
</tr>
<tr>
<td>open/close</td>
<td>80%</td>
<td>18%</td>
</tr>
<tr>
<td>fork</td>
<td>6%</td>
<td>2%</td>
</tr>
<tr>
<td>exec</td>
<td>3%</td>
<td>&lt;1%</td>
</tr>
</tbody>
</table>

Macro-benchmark

<table>
<thead>
<tr>
<th>Prog.</th>
<th>Whole</th>
<th>Selective</th>
</tr>
</thead>
<tbody>
<tr>
<td>unpack</td>
<td>2%</td>
<td>&lt;1%</td>
</tr>
<tr>
<td>build</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>postmark</td>
<td>11%</td>
<td>6%</td>
</tr>
</tbody>
</table>

Selective: cost of allocating/freeing provenance “blob” + recording or not decision

Whole: Selective + cost of recording provenance information

More details in the paper
Future work

- Bind with existing storage/query/processing tools
  - Ongoing collaboration with MIT Lincoln team
- Focusing on building provenance applications
  - Do we capture everything we need?
  - Is usability good enough?
- Improving tooling for container support
  - Doable, but requires to dive in
- Providing some (formal?) guarantee of completeness
- Extending user-base and providing support
- Programmable queries
- Submitting a patch for integration in mainline kernel?
Thank you!

Any questions?

tfjmp@seas.harvard.edu

@tfjmp

tfjmp.org

Source, demo and benchmark at:

www.camflow.org