Decoupling Provenance Capture and Analysis from Execution

Manolis Stamatogiannakis (@mstamat)
Paul Groth (@pgroth)
Herbert Bos

Capturing Provenance

Disclosed Provenance

+ Accuracy
+ High-level semantics
- Intrusive
- Manual Effort

CPL (Macko ‘12)
Trio (Widom ‘09)
PrIME (Miles ‘09)
Taverna (Oinn ‘06)
VisTrails (Fraire ‘06)

Observed Provenance

- False positives
- Semantic Gap
+ Non-intrusive
+ Minimal manual effort

ES3 (Frew ‘08)
Trec (Vahdat ‘98)
PASSv2 (Holland ‘08)
DTrace Tool (Gessiou ‘12)
What to capture?

Provenance is post hoc.

Aim:
Eliminate the need for developers to know what provenance needs to be captured.
Re-execution

Common tactic in provenance:

• **DB**: Reenactment queries (Glavic ‘14)
• **DistSys**: Chimera (Foster ‘02), Hadoop (Logothetis ‘13), DistTape (Zhao ‘12)
• **Workflows**: Pegasus (Groth ‘09)
• **PL**: Slicing (Perera ‘12)
• **OS**: pTrace (Guo ‘11)
• **Desktop**: Excel (Asuncion ‘11)
RECORD
/dev/input/event*
Record input events
Event Trace

REPLAY
Event Trace
Replay Agent
Inject Events
/dev/input/event*
Methodology

Execution Capture

Instrumentation

Provenance analysis

Selection
Prototype Implementation (1/2)

- PANDA: an open-source Platform for Architecture-Neutral Dynamic Analysis. (Dolan-Gavitt et al. ‘14)
- Based on QEMU virtualization platform.
- Logs self-contained execution traces.
  - An initial RAM snapshot.
  - Non-deterministic inputs.

Diagram:
- PANDA
  - Input
  - Interrupt
  - Initial RAM Snapshot
  - Non-determinism log
  - DMA
  - PANDA Execution Trace
Prototype Implementation (2/2)

- Debian Linux guest.
- Analysis plugins
  - Read-only access to the VM state.
  - Invoked per instr., memory access, context switch, etc.
  - Can be combined to implement complex functionality.
  - OSI Linux, PROV-Tracer, ProcStrMatch.
- Provenance stored PROV/RDF triples, queried with SPARQL.
OS Introspection

• What processes are currently executing?
• Which libraries are used?
• What files are used?

• Possible approaches:
  – Execute code inside the guest OS.
  – Reproduce guest-OS semantics purely from the hardware state (RAM/registers).
Introspecting Kernel Structures (1/2)

- Kernel structure members are known.
- Their offsets depend on compile-time configuration.
- Each Linux vendor supplies a few different kernel configurations.
- Rule of thumb: same vendor/configuration/version combo $\rightarrow$ same offsets.
Offset profile created once for each kernel “family”.

E.g. one profile for all Debian/amd64/3.2.* kernels.

The profile is used by osi_linux module to extract process info from the execution trace.

```c
struct task_struct t;
int off = &t.memb - &t;
printk("off: %d", off);
```
The PROV-Tracer Plugin

- Registers for process creation/destruction events.
- Decodes executed system calls.
- Keeps track of what files are used as input/output by each process.
- Emits provenance in an intermediate format when a process terminates.
More Analysis Plugins

• ProcStrMatch plugin.
  – Which processes contained string S in their memory?

• Other possible types of analysis:
  – Taint tracking
  – Dynamic slicing
Execution Overhead (1/2)

• QEMU incurs a 5x slowdown.
• PANDA recording imposes an additional 1.1x – 1.2x slowdown.

Virtualization is the dominant overhead factor.
Execution Overhead (2/2)

• QEMU is a suboptimal virtualization option.
• ReVirt – User Mode Linux (Dunlap et al. ‘02)
  – Slowdown: 1.08x rec. + 1.58x virt.
• ReTrace – VMWare (Xu et al. ‘07)
  – Slowdown: 1.05x-2.6x rec. + ??? virt.

Virtualization slowdown is considered acceptable.
Recording overhead is fairly low.
Storage Requirements

• Storage requirements vary with the workload.
• For PANDA (Dolan-Gavitt et al. ’14):
  – 17-915 instructions per byte.
• In practice: O(10MB/min) uncompressed.
• Different approaches to reduce/manage storage requirements.
  – Compression, HD rotation, VM snapshots.
• 24/7 recording seems within limits of today’s technology.
An Example (1)

$ qemu -replay alice.et -panda
  "osi;osi_linux;prov_tracer"
$ ./raw2ttl.py < prov_out.raw > alice.ttl

<exe://pam-foreground--~3451> prov:endedAtTime 199090196.
<exe://getent~3451> a prov:Activity.
<exe://cut~3452> a prov:Activity.
<file:/etc/nsswitch.conf> a prov:Entity.
<file:/etc/nsswitch.conf> rdfs:label "/etc/nsswitch.conf".
<file:/etc/nsswitch.conf> rdf:type dt:Unknown.
<exe://getent~3451> prov:used <file:/etc/nsswitch.conf> .
# unused file:3477815296:getent~3451:/etc/passwd:r0:w0:f524288
<exe://getent~3451> prov:startedAtTime 199090196.
<exe://getent~3451> prov:endedAtTime 200392668.
<file:FD0_3452> a prov:Entity.
<file:FD0_3452> rdfs:label "FD0_3452"
An Example (2)

```sql
SELECT (MIN(?startTime) AS ?s)
  (MAX(?endTime) AS ?e) WHERE
{
  <file:/home/panda/work/camelidae-new.tar>
  prov:wasDerivedFrom ?file .
  FILTER regex(str(?file), "txt")

  ?file prov:wasGeneratedBy ?activity .
  ?activity a dt:Editor .
}

Results:

<table>
<thead>
<tr>
<th>s</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>705412095</td>
<td>990055363</td>
</tr>
</tbody>
</table>
Example (3)

```bash
$ export s=705412095 e=990055363
$ qemu -replay alice.et -panda
   "scissors:start=$s,end=$e,name=report.et"

$ echo ""Llamas are lame"" > search_strings.txt
$ qemu -replay report.et -panda
   "osi;osi_linux;callstack_instr;stringsearch;psstrmatch"
```
Example (4)

```sql
SELECT (MIN(?st) as ?min) ?vi WHERE {
    ?vi dt:hasMemText "Llamas are lame".
    ?vi prov:startedAtTime ?st
} GROUP BY ?vi
```

Results:

<table>
<thead>
<tr>
<th>min</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>705412095</td>
<td>exe://vi~3547</td>
</tr>
<tr>
<td>782648505</td>
<td>exe://vi~3557</td>
</tr>
<tr>
<td>857809758</td>
<td>exe://vi~3570</td>
</tr>
<tr>
<td>963886071</td>
<td>exe://vi~3595</td>
</tr>
</tbody>
</table>
Conclusion

• Decouple capture/analysis from execution
• VMs provide a useful indirection mechanism
• Future:
  – More plugins
  – Real world analysis (rrshare.org)
  – Cloud analysis
• Are traces/immutable logs primitive?
Source & Text

• PROV-Tracer source:
  – https://github.com/m000/panda/tree/prov_tracer/
  – Plugins under qemu/panda_plugins.

• Full text of paper: