Provenance-based Intrusion Detection

Thomas Pasquier, University of Bristol

UK Cyber Security Winter School, Newcastle, 15/01/2020
Talk loosely based on the following publications

- Han et al. “UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats”, NDSS 2020
- Pasquier et al. “Runtime Analysis of Whole-System Provenance”, CCS 2018
- Pasquier et al. “Practical Whole-System Provenance Capture”, SoCC 2017
System call based intrusion detection

System Calls
System call based intrusion detection

System Calls

Identify abnormal patterns
System call based intrusion detection

System Calls

Identify abnormal patterns

Hidden among benign actions
System call based intrusion detection

System Calls

Identify abnormal patterns

Hidden among benign actions
Masquerading as benign action
System call based intrusion detection

System Calls

[...] Identify abnormal patterns

Hidden among benign actions Masquerading as benign action

Over a long period of time
How to solve this with provenance?

- What is provenance?
- Why use provenance?
- How to capture provenance?
- How to perform detection?
- How to evaluate?
- Insights
What is provenance?
System-level provenance graph

- History of a system execution
- Represent interactions between system objects
- Represented as a **directed acyclic graph**
- Information Flows
- **Relationship** between kernel object states
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Why use provenance?
Provenance-based intrusion detection

- **Intuition**: provenance graph exposes causality relationships between events
Provenance-based intrusion detection

- **Intuition**: provenance graph exposes causality relationships between events
Provenance-based intrusion detection

- Related events are connected even across long period of time
How to capture provenance?
Capture methods

Examples

3. Pasquier et al. "Practical whole-system provenance capture" SoCC. 2017
Capture methods

Examples

3. Pasquier et al. "Practical whole-system provenance capture" SoCC. 2017
Why are they not appropriate?
Interposition is unsafe

- Watson "Exploiting Concurrency Vulnerabilities in System Call Wrappers" WOOT. 2007

- **Time-of-audit-to-time-of-use attack**
  - Race condition

- **Syntactic Race**
  - different copy of parameters

- **Semantic Race**
  - Kernel state may change
Capture methods

Examples

1. Based on Linux reference monitor
2. Best accuracy
3. Stronger formal guarantees
4. Formally specified semantic
5. Best performance

Pasquier et al. “Runtime Analysis of Whole-System Provenance”, CCS 2018
How to perform detection?
Assumptions (and limitations)

- **Runtime detection**
  - We target environment with *minimal human intervention*
    - relatively consistent behaviour
    - e.g. web servers, CI pipelines etc...
  - Build a **model of system behaviour** (unsupervised training)
    - in a controlled environment
    - from a representative workload (this is hard!)

- **Detect deviation** from the model
- Several approaches being explored…
Example
Example

1) Graph streamed in, converted to histogram, labelled using (modified) **struct2vec**
2) At regular interval, histogram converted to a fixed size vector using similarity preserving hashing
3) Feature vectors are **clustered**.
4) Cluster forms "meta-state", transitions are modelled

In deployment, anomaly detected via clustering and "meta-state" model
Relatively simple

- Nothing overly fancy here
- Labelled directed acyclic graph
  - node/edge types
  - security context (when available)
- Modification and combination of existing algorithms
  - struct2vec
  - similarity preserving hashing
  - clustering
- Right combination + domain knowledge
How to evaluate?
Comparison state of the art


<table>
<thead>
<tr>
<th>Experiment</th>
<th>Dataset</th>
<th># of Graphs</th>
<th>Avg.</th>
<th>Avg.</th>
<th>Preprocessed Data Size (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YouTube</td>
<td>100</td>
<td>8,292</td>
<td>113,239</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>Gmail</td>
<td>100</td>
<td>6,827</td>
<td>37,382</td>
<td>0.1</td>
</tr>
<tr>
<td>StreamSpot</td>
<td>Download</td>
<td>100</td>
<td>8,831</td>
<td>310,814</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>VGame</td>
<td>100</td>
<td>8,637</td>
<td>112,958</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>CNN</td>
<td>100</td>
<td>8,990</td>
<td>294,903</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>Attack</td>
<td>100</td>
<td>8,891</td>
<td>28,423</td>
<td>0.1</td>
</tr>
</tbody>
</table>

TABLE I: Characteristics of the StreamSpot dataset. The dataset is publicly available only in a preprocessed format.

R -> neighborhood size for struct2vec algorithm

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamSpot (baseline)</td>
<td>0.74</td>
<td>N/A</td>
<td>0.66</td>
<td>N/A</td>
</tr>
<tr>
<td>$R = 1$</td>
<td>0.51</td>
<td>1.0</td>
<td>0.60</td>
<td>0.68</td>
</tr>
<tr>
<td>$R = 3$</td>
<td>0.98</td>
<td>0.93</td>
<td>0.96</td>
<td>0.94</td>
</tr>
</tbody>
</table>

TABLE II: Comparison to StreamSpot on the StreamSpot dataset. We estimate StreamSpot’s average accuracy and precision from the figure included in the paper [83], which does not report exact values. They did not report recall or F-score.
Evaluation with DARPA datasets

| Experiment | Dataset   | # of Graphs | Avg. $|V|$ | Avg. $|E|$ | Raw Data Size (GiB) |
|------------|-----------|-------------|-------|-------|---------------------|
| DARPA      | Benign    | 66          | 59,983| 4,811,836 | 271                 |
| CADETS     | Attack    | 8           | 386,548| 5,160,963 | 38                  |
| DARPA      | Benign    | 43          | 2,309 | 4,199,309 | 441                 |
| ClearScope | Attack    | 51          | 11,769| 4,273,003 | 432                 |
| DARPA      | Benign    | 2           | 19,461| 1,913,202 | 4                   |
| THEIA      | Attack    | 25          | 275,822| 4,073,621 | 85                  |

TABLE IV: Characteristics of graph datasets used in the DARPA experiments.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARPA CADETS</td>
<td>0.98</td>
<td>1.0</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>DARPA ClearScope</td>
<td>0.98</td>
<td>1.0</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>DARPA THEIA</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

TABLE V: Experimental results of the DARPA datasets.
Evaluation with DARPA datasets

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Dataset</th>
<th># of Graphs</th>
<th>Avg.</th>
<th>Avg.</th>
<th>Raw Data Size (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARPA</td>
<td>Benign</td>
<td>66</td>
<td>59,983</td>
<td>4,811,836</td>
<td>271</td>
</tr>
<tr>
<td>CADETS</td>
<td>Attack</td>
<td>8</td>
<td>386,548</td>
<td>5,160,963</td>
<td>38</td>
</tr>
<tr>
<td>DARPA</td>
<td>Benign</td>
<td>43</td>
<td>2,309</td>
<td>4,199,309</td>
<td>441</td>
</tr>
<tr>
<td>ClearScope</td>
<td>Attack</td>
<td>51</td>
<td>11,769</td>
<td>4,273,003</td>
<td>432</td>
</tr>
<tr>
<td>DARPA</td>
<td>Benign</td>
<td>2</td>
<td>19,461</td>
<td>1,913,202</td>
<td>4</td>
</tr>
<tr>
<td>THEIA</td>
<td>Attack</td>
<td>25</td>
<td>275,822</td>
<td>4,073,621</td>
<td>85</td>
</tr>
</tbody>
</table>

TABLE IV: Characteristics of graph datasets used in the DARPA experiments.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARPA CADETS</td>
<td>0.98</td>
<td>1.0</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>DARPA ClearScope</td>
<td>0.98</td>
<td>1.0</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>DARPA THEIA</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

TABLE V: Experimental results of the DARPA datasets.
Building our own dataset

- Attack designed to look similar to background activity

| Experiment | Dataset | # of Graphs | Avg. | | Avg. | | Raw Data Size (GiB) |
|------------|---------|-------------|------|------|------|---------------------|
| SC-1       | Benign  | 125         | 265,424 | 975,226 | 64 |
|            | Attack  | 25          | 257,156 | 957,968 | 12 |
| SC-2       | Benign  | 125         | 238,338 | 911,153 | 59 |
|            | Attack  | 25          | 243,658 | 949,887 | 12 |

TABLE VI: Characteristics of the datasets used in the supply-chain APT attack experiments.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC-1</td>
<td>0.85</td>
<td>0.96</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>SC-2</td>
<td>0.75</td>
<td>0.80</td>
<td>0.77</td>
<td>0.78</td>
</tr>
</tbody>
</table>

TABLE VIII: Experimental results of the supply-chain APT attack scenarios.
Runtime performance

Fig. 4: Total number of processed edges over time (in seconds) in the SC-1 experimental workload with varying batch sizes (Fig. 4(a)), sketch sizes (Fig. 4(b)), hop counts (Fig. 4(c)), and intervals of sketch generation (Fig. 4(d)). Dashed blue line represents the speed of graph edges streamed into UNICORN for analysis. Triangle showcase baseline has the same configurations as those used in our experiments and indicates the values of the controlled parameters (that remain constant) in each figure.

F. CPU & Memory Utilization

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Parameter Value</th>
<th>Max Memory Usage (MBs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H = 3</td>
<td></td>
<td>947</td>
</tr>
</tbody>
</table>
Runtime performance

Memory usage: ~500MB
CPU usage 15% on 1 core

F. CPU & Memory Utilization

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Parameter Value</th>
<th>Max Memory Usage (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H = 3</td>
<td></td>
<td>945</td>
</tr>
</tbody>
</table>
Insights
Provenane-based IDS work!

- We can detect intrusion out of graph structure with little metadata
  - Vertex type (thread, file, socket etc…)
  - Edge type (read, write, connect etc…)
- Processing speed
  - Current prototype
  - Data generation speed < processing speed!
Proper evaluation is hard!

- Dataset are hard to generate
  - What is a good quality dataset?
- Hard to compare across papers, a lot is not available
  - Experiments (i.e. attacks)
  - Capture Mechanisms
  - Analysis pipelines
- Leads to unsatisfactory evaluation
  - I may be able to compare to similar techniques (may reuse dataset)
  - … very hard for unrelated one (i.e. ingest different data type)
Explainability is a problem

- There is a problem within the last batch of X graph elements
  - 2,000 in previous figures
- Good luck finding out what went wrong
- Provenance forensic is an active field of research
  - Promising work out of the DARPA programme
- ... but could we do better during detection?
  - Promising work with colleagues at NEC Labs America
  - Report vertex within 3 nodes of anomaly in 75% of cases!
  - Deep graph learning techniques
What about unpredictable workload?
Everything has been open-sourced

- Capture: [http://camflow.org](http://camflow.org)
  - Linux package available!
- IDS: [https://github.com/crimson-unicorn](https://github.com/crimson-unicorn)
- Streamspot data: [https://github.com/sbustreamspot/sbustreamspot-data](https://github.com/sbustreamspot/sbustreamspot-data)
- DARPA data: [https://github.com/darpa-i2o/Transparent-Computing](https://github.com/darpa-i2o/Transparent-Computing)

For those really interested (20+ papers on the topic):
- [https://github.com/tfjmp/provenance-papers](https://github.com/tfjmp/provenance-papers)
Thank you, questions?
Tfjmp.org
thomas.pasquier@bristol.ac.uk