Towards provenance-based intrusion detection
Thomas Pasquier, University of Bristol

Workshop on Machine Learning for Cyber Security
Loughborough University, 11/03/2019
Talk loosely based on following publications

- Han et al. “UNICORN: Revisiting Host-Based Intrusion Detection in the Age of Data Provenance”, working paper
- Pasquier et al. “Runtime Analysis of Whole-System Provenance”, ACM CCS 2018
- Han et al. “Provenance-based Intrusion Detection: Opportunities and Challenges”, USENIX TaPP 2018
- Han et al. “FRAPpuccino: Fault-detection through Runtime Analysis of Provenance”, USENIX HotCloud 2017
- Pasquier et al. “Practical Whole-System Provenance Capture”, ACM SoCC 2017
System call based intrusion detection

System Calls

bristol.ac.uk
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
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 system states are connected even across long period of time
What is provenance in an operating system?

- Represent interactions between system objects
- Represented as a **directed acyclic graph**
- Information Flows
- **Relationship** between kernel object states
- History of a system execution
Example provenance
Provenance-based Intrusion Detection

- We target cloud application
  - Relatively well defined behaviour
- Build a model of system behaviour (unsupervised, batch training)
  - in a controlled environment
  - from a representative workload
  - assumed as part of the CI toolchain
- Detect deviation from the model
- Several approaches being explored…
Detecting intrusion
Detecting intrusion

1) Graph streamed in, converted to histogram, labelled using struct2vec
Detecting intrusion

2) At regular interval, histogram converted to a fixed size vector using similarity preserving hashing
Detecting intrusion

3) Feature vectors are clustered
4) Cluster forms “meta-state”, transitions are modelled
In deployment, anomaly detected via clustering and “meta-state” model
How well does it work?

<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</td>
<td>0.98/[0.48-1.0]</td>
<td>0.93</td>
<td>0.96/[0.50-0.82]</td>
<td>0.94</td>
</tr>
<tr>
<td>DARPA Cadets</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>wget Baseline</td>
<td>1.0</td>
<td>0.88</td>
<td>0.91</td>
<td>0.94</td>
</tr>
<tr>
<td>wget Interval</td>
<td>1.0</td>
<td>0.84</td>
<td>0.89</td>
<td>0.91</td>
</tr>
</tbody>
</table>

TABLE IV: Experimental results. We estimate StreamSpot’s accuracy and average precision (in [red]) from the figure included in the paper, which does not report exact values. They did not report recall or F-score.
How well does it work?

Fig. 3: Total number of processed edges over time (in seconds) of UNICORN in a wget experimental workload with varying batch sizes (left), sketch sizes (middle left), hop counts (middle right), and intervals of sketch generation (right). Dashed blue line represents the speed of graph edges streamed into UNICORN for analysis. Red baseline has the same configurations as those used in our experiment and indicates the values of the controlled parameters (that remain constant) in each figure.

Fig. 4: Detection performance (precision, recall, accuracy, and F-score) with varying hop counts (left), sketch sizes (middle), and intervals of sketch generation (right). Baseline values are used by the controlled parameters (that remain constant) in each figure.
Some insights

- 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!
Some insights

- Doing 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
Research Trajectory

- Capture system open-sourced ([http://camflow.org](http://camflow.org))
  - Maintained for ~4 years now
  - Used in multiple research projects
  - Datasets soon to be released
  - Exploring means to perform cross-evaluation in an evolving “capture field”
- Extending to distributed systems
  - Capture system already support this
- Exploring more advanced ML techniques
  - Although the relatively simple approach presented bring good results
Thank you, questions?

tfjmp.org
camflow.org