Building a provenance-based intrusion detection system

Thomas Pasquier, University of Bristol
AMSS, 08/12/2020
Talk loosely based on following publications

- Han et al. “UNICORN: Revisiting Host-Based Intrusion Detection in the Age of Data Provenance”, NDSS 2020
- Pasquier et al. “Runtime Analysis of Whole-System Provenance”, ACM CCS 2018
- Pasquier et al. “Practical Whole-System Provenance Capture”, ACM SoCC 2017

https://tfjmp.org
Plan: Building a provenance-based intrusion detection system

- Motivation
- What is provenance?
- How is useful?
- How to perform detection?
- How to evaluate
- Some insights from this work
Motivation
Motivation: System call based intrusion detection

System Calls
Motivation: System call based intrusion detection

System Calls

Identify abnormal patterns
Motivation: System call based intrusion detection

System Calls

Identify abnormal patterns
Hidden among benign actions
Motivation: System call based intrusion detection

System Calls

Identify abnormal patterns

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

System Calls

- Identify abnormal patterns
- Hidden among benign actions
- Masquerading as benign action
- Over a long period of time
What is provenance?
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 provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)
Example provenance (simplified)

Linux kernel compilation:
~2M graph elements
How is this useful?
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 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: UNICORN

- Han et al. “UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats”, NDSS 2020
Example: UNICORN

1) Graph streamed in, converted to histogram, labelled using (modified) \textit{struct2vec}
Example: UNICORN

2) At regular interval, histogram converted to a fixed size vector using similarity preserving graph sketching
Example: UNICORN

3) Feature vectors are **clustered**
Example: UNICORN

4) Cluster forms “meta-state”, transitions are modelled
In deployment, anomaly detected via clustering and “meta-state” model
Relatively simple

- 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?
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.
Evaluation with DARPA datasets

SUCH GOOD RESULTS ARE NOT NORMAL

<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

<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>SC-1</td>
<td>Benign</td>
<td>125</td>
<td>265,424</td>
<td>975,226</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Attack</td>
<td>25</td>
<td>257,156</td>
<td>957,968</td>
<td>12</td>
</tr>
<tr>
<td>SC-2</td>
<td>Benign</td>
<td>125</td>
<td>238,338</td>
<td>911,153</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Attack</td>
<td>25</td>
<td>243,658</td>
<td>949,887</td>
<td>12</td>
</tr>
</tbody>
</table>

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.
Building our own dataset

- Attack designed to look similar to background activity
- Is that enough?

| Experiment | Dataset | # of Graphs | Avg. $|V|$ | Avg. $|E|$ | 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.
Some insights from this work
We can build practical provenance-based IDSs

- 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)
- Adversarial ML?
Identifying threats: 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?
  - Graph LSTM Autoencoder
  - Pinpoint issue in the graph, but...
  - ...less complex problem
  - ...smaller graphs
  - Can we take such approach further?
Looking for ...

- PhD Students
- Postdocs: job advert to come (soon) on intrusion detection at the edge
- Collaborators

Get in touch: https://tfjmp.org
Thank you! Questions?

tfjmp.org
camflow.org
Comparison state of the art

Manzoor et al. "Fast memory-efficient anomaly detection in streaming heterogeneous graphs"
ACM KDD, 2016.

R -> neighborhood size for struct2vec algorithm
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 maxon 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 (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H = 1</td>
<td></td>
<td>9457</td>
</tr>
</tbody>
</table>
Runtime performance

![Graph showing runtime performance with different hop counts](image)
Runtime performance

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

- 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

Modifications to the Linux Kernel code

<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 (v5.4.15)</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>4220</td>
</tr>
<tr>
<td>circa 2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Capture 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>

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

Whole: Selective + cost of recording provenance information

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>
IDS performance (more)

Figure 4: Detection performance (precision, recall, accuracy, and F-score) with varying hop counts (Fig. 4a), sketch sizes (Fig. 4b), intervals of sketch generation (Fig. 4c), and decay factor (Fig. 4d). Baseline values (*) are used by the controlled parameters (that remain constant) in each figure.
IDS performance (more)

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Parameter Value</th>
<th>Max Memory Usage (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hop Count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R = 1</td>
<td></td>
<td>562</td>
</tr>
<tr>
<td>R = 2</td>
<td></td>
<td>624</td>
</tr>
<tr>
<td>R = 3</td>
<td></td>
<td>687</td>
</tr>
<tr>
<td>R = 4</td>
<td></td>
<td>749</td>
</tr>
<tr>
<td>R = 5</td>
<td></td>
<td>812</td>
</tr>
<tr>
<td>Sketch Size</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>312</td>
</tr>
<tr>
<td></td>
<td></td>
<td>437</td>
</tr>
<tr>
<td></td>
<td></td>
<td>687</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,374</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2,498</td>
</tr>
</tbody>
</table>

Table 5: Memory usage with varying hop counts and sketch sizes.

Figure 6: Per virtual CPU and average CPU utilization.
IDS performance (more)

CPU over long time period? 15% CPU time across cores

Figure 5: Average CPU utilization with the baseline configurations.
Advanced Persistent Threats

1. Reconnaissance
   - Identify Target & Explore Vulnerabilities
2. Weaponize
   - Design Backdoor & Penetration Plan
3. Delivery
   - Deliver the Weapon
4. Exploitation
   - Victim Triggers Vulnerability
5. Installation
   - Install Backdoor or Malware
6. Command & Control
   - Give Remote Instructions to Victim
7. Actions on Objectives

Diverse Attack Vectors
- Active Scanning
- Passive Scanning
- Malware
- Scripting
- Spearphishing
- Supply-chain Attack
- Application Shimming
- Job Scheduling
- Hooking
- Dylib Hijacking
- Connection Proxy
- Domain Fronting

Zero-Day Exploits

Long Duration → Low-and-Slow Attack Patterns