FRAPpuccino: Fault-detection through Runtime Analysis of Provenance

Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein and Margo Seltzer

Harvard University
Motivations

• PaaS clouds are popular and the market continues to grow (~30% annually)
  – But cloud security remains challenging.
• Cloud applications can serve millions of users
  – Run-time faults can render the service unavailable.
• It would be nice to have an automated detection system with high accuracy and no application annotation effort.
FRAP in One Slide
Outline

• Background: what is provenance?
• Model generation
• Detection algorithm
• Experimental results
• Conclusions
• Discussion Topics
Provenance (1)

- Provenance tracks the chronology of objects/resources.
- Whole-system provenance records a program’s activities on the host system.
  - Example: Alice creates a file `a.txt`.
Provenance (2)

Interactions between a program and its host OS naturally form a DAG.

| W3C PROV Data Model Type | DAG Representation | Example |
|--------------------------|--------------------|---------|
| Entity/Activity          | Node               | Kernel data objects (e.g., files, packets) Inode attributes, network addresses, etc. |
| Relationship             | Edge               | Processes manipulate entities |
| Agent                    | Node               | Users and groups that enact activities |

![Diagram showing relationships between entities and activities in a DAG]

- **Executable** wasAssociatedWith **Process**
- **Input file** used as **Process**
- **Process** wasGeneratedBy **Output file**
- **All** used **Executable** and **Input file**
Model Generation

• Determine the size of provenance data that captures program behavior $\rightarrow$ **dynamic sliding window**
• Generate a **feature vector** from each provenance DAG.
• Clustering FVs to create a program model
  – **Centroid** of each cluster
  – Cluster **radii**
  – **Membership** of each cluster
• Isolated FVs are discarded
Dynamic Sliding Window

- A subset of unbounded provenance data can describe normal program behavior
- *Dynamic*: determine the window size based on the provenance records during program run
- *Sliding*: continuously monitor different subsets of provenance data during detection
Feature Vector

- Projection of a DAG as a point into an $n$-dimensional space
- Contains counts of DAG labels
- Labels encode program interactions with the system
Generating Feature Vector: 1st Iteration

In: 1, 2a2b

| Label String | New Label |
|--------------|-----------|
| 1, 2a2b      | 4         |
Generating Feature Vector: 1st Iteration

In: 1, 2a2b
Out: 1, NULL

| Label String | New Label |
|--------------|-----------|
| 1, 2a2b      | 4         |
| 1, NULL      | 5         |
| 4, 5         | 6         |
Generating Feature Vector: 1\textsuperscript{st} Iteration

| Label String | New Label |
|--------------|-----------|
| 1, 2a2b      | 4         |
| 1, NULL      | 5         |
| 4, 5         | 6         |
| 2, 3b        | 7         |
| 2, 1a2c      | 8         |
| 7, 8         | 9         |

In: 1, 2a2b
Out: 1, NULL

In: 2, 3b
Out: 2, 1a2c
Generating Feature Vector: 1\textsuperscript{st} Iteration

| Label String          | New Label |
|-----------------------|-----------|
| 1, 2a2b               | 4         |
| 1, NULL               | 5         |
| 4, 5                  | 6         |
| 2, 3b                 | 7         |
| 2, 1a2c               | 8         |
| 7, 8                  | 9         |
| 2, 2c3b               | 10        |
| 2, 1b                 | 11        |
| 10, 11                | 12        |
Generating Feature Vector: 1\textsuperscript{st} Iteration

| Label String | New Label |
|--------------|-----------|
| 1, 2a2b      | 4         |
| 1, NULL      | 5         |
| 4, 5         | 6         |
| 2, 3b        | 7         |
| 2, 1a2c      | 8         |
| 7, 8         | 9         |
| 2, 2c3b      | 10        |
| 2, 1b        | 11        |
| 10, 11       | 12        |
| 3, NULL      | 13        |
| 3, 2b        | 14        |
| 13, 14       | 15        |
Generating Feature Vector: 1\textsuperscript{st} Iteration

\begin{tabular}{|c|c|}
\hline
Label String & New Label \\
\hline
1, 2a2b & 4 \\
1, NULL & 5 \\
4, 5 & 6 \\
2, 3b & 7 \\
2, 1a2c & 8 \\
7, 8 & 9 \\
2, 2c3b & 10 \\
2, 1b & 11 \\
10, 11 & 12 \\
3, NULL & 13 \\
3, 2b & 14 \\
13, 14 & 15 \\
\hline
\end{tabular}
Generating Feature Vector: 1\textsuperscript{st} Iteration

| Label String   | New Label |
|----------------|-----------|
| 1, 2a2b       | 4         |
| 1, NULL       | 5         |
| 4, 5          | 6         |
| 2, 3b         | 7         |
| 2, 1a2c       | 8         |
| 7, 8          | 9         |
| 2, 2c3b       | 10        |
| 2, 1b         | 11        |
| 10, 11        | 12        |
| 3, NULL       | 13        |
| 3, 2b         | 14        |
| 13, 14        | 15        |
Feature Vector After 1st Iteration

\[ 1\ 2\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 2 \]
Clustering FVs

• K-means clustering of all feature vectors
  – Determine K by clustering pairwise distances
  – Counts are transformed to probability distributions if needed

• Experiment with distance metrics
  – Kullback-Leibler with back-off probability
  – Hellinger
  – Euclidean
Detection Algorithm (1)

The diagram illustrates a process involving PaaS (Platform as a Service), which feeds into a network structure. This network is then subjected to FV (Feature Vector) Generation. The output from FV Generation is fed into Model Fitting, which results in a decision between Normal and Abnormal conditions. The specific data points (01001 02304) suggest a particular analysis or threshold used in the model fitting process.
Detection Algorithm (2)

- Continuously monitor a running instance using the dynamic sliding window
- Only store and analyze provenance data within the window

![Example Detection Algorithm (Window Size = 4)](#)

**Learning the Model**

**Using the Model**
Experiment Setup

- Ruby server out-of-memory crash
- Faulty server code causes out-of-memory crash when a client requests a particular URL.
- FRAP monitors many instances of a Ruby Server, modeling its normal behavior.
### Experimental Results

| Distance Metrics | Isolate Bad Instance During Model Generation? | Captured Bad Instance During Continuous Detection? |
|------------------|---------------------------------------------|--------------------------------------------------|
| Kullback-Leibler  | ✔                                          | ✔                                                |
| Hellinger        | ❌                                          | ❌                                                |
| Euclidean        | ✔                                          | ✔                                                |

- Experiment uses 10 server instances accepting client requests
- 1 instance crashes during model generation
- The same instance crashes again during detection
Conclusions

• Security is still a major concern of the PaaS clouds.
• Provenance provides an alternative approach to detecting faults/intrusions.
• Preliminary experiments show promising results of such an approach.
• Multiple exciting future directions exist.
  – Incorporating more machine learning algorithms?
  – Provenance database of known vulnerabilities?
  – Differential provenance?
Discussion Topics

• What if provenance data are not trustworthy? Can we integrate detection of provenance data tampering?
• How can we use provenance to provide meaningful information to the users when an intrusion is detected?
• What are the pros and cons of FRAP compared to other behavioral-based detection systems and to the cloud IDS’s at large?