FRAPpuccino:
Fault-detection through Runtime Analysis of Provenance

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

Harvard University

Abstract

We present FRAPpuccino (or FRAP), a provenance-based fault detection mechanism for Platform as a Service (PaaS) users, who run many instances of an application on a large cluster of machines. FRAP models, records, and analyzes the behavior of an application and its impact on the system as a directed acyclic provenance graph. It assumes that most instances behave normally and uses their behavior to construct a model of *legitimate behavior*. Given a model of legitimate behavior, FRAP uses a *dynamic sliding window* algorithm to compare a new instance's execution to that of the model. Any instance that does not conform to the model is identified as an anomaly. We present the FRAP prototype and experimental results showing that it can accurately detect application anomalies.

System Overview

FRAP models a PaaS application's normal behavior using provenance DAGs of its many running instances. Provenance DAGs are converted into *feature vectors* (FVs) that constitute the model. Once a model is constructed, it monitors the application by continuously analyzing its provenance DAG and fitting its FV against the model. FRAP notifies the user when it detects behavior that deviates from the model. This process is illustrated in Figure 1.

Provenance DAGs

Whole-system provenance tracks a program's activities on the host system. The interactions naturally form a DAG with nodes representing system-level *entities* (e.g., files) and *agents* (e.g., users), and edges representing *activities* (e.g., used). Figure 3 shows an example.

Model Generation

FRAP takes the provenance DAG within a *dynamic sliding window* (DSW) and generates a FV for each instance. DSWs enable FRAP to process streaming provenance data, while storing and analyzing only those parts that capture the important aspect of program behavior. A FV is a projection of a DAG into an n-dimensional space. It is generated through a variation of a label propagation algorithm based on the subtree Weisfeiler-Lehman graph kernel. FRAP clusters FVs to create a program model, which contains:

– **Centroid** of each cluster;
– Cluster **radii**;
– **Membership** of each cluster.

Isolated FVs are discarded.

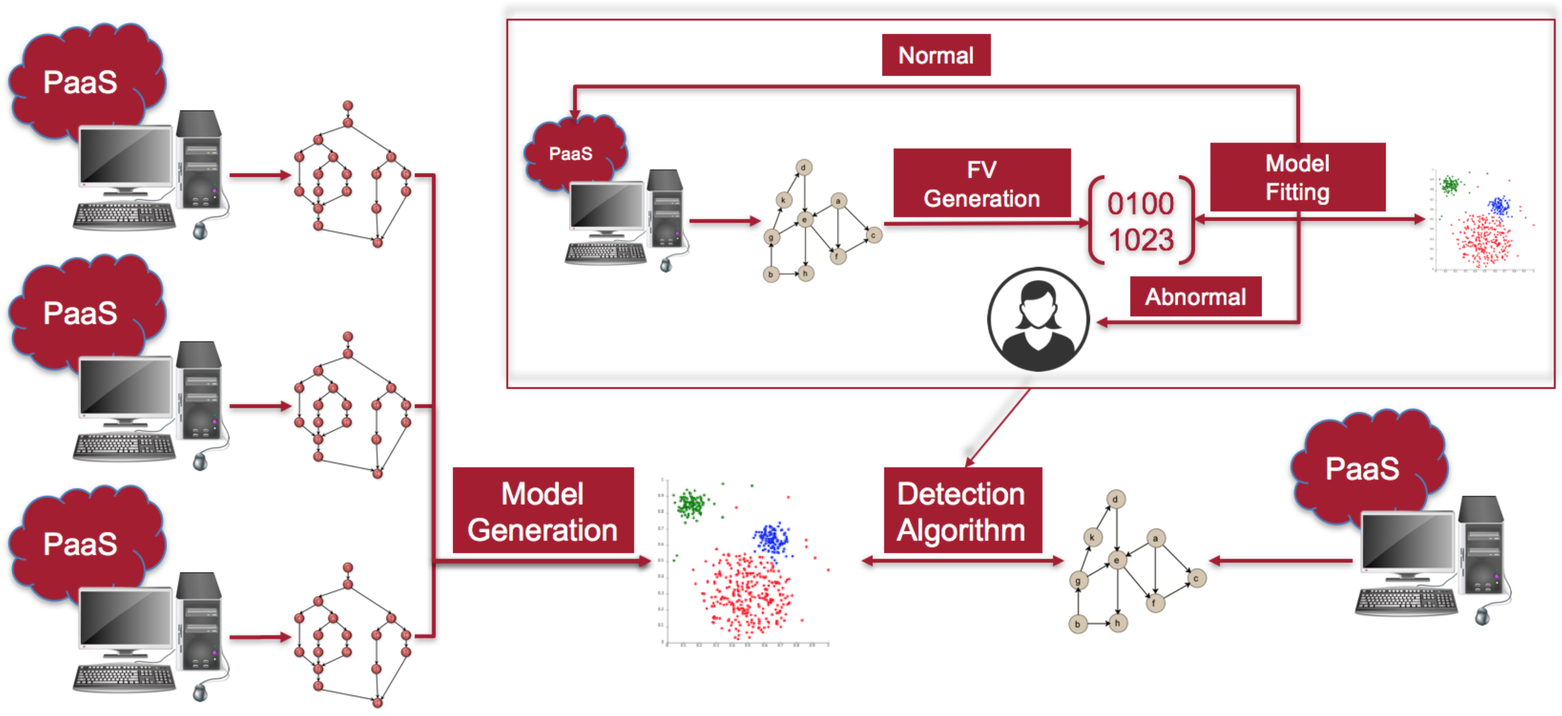

Figure 1: FRAP's framework and the workflow of its detection algorithm at a glance.

Detection Algorithm

Once a model is generated, FRAP continues to monitor the program as it slides the DSW and analyzes the DAG within the window. FRAP generates a FV using the DAG and fits it into the model, as illustrated in the callout box in Figure 1. Normal behavior can either be fitted into the radius of a cluster or be reclustered with the members of a cluster. If the FV is isolated, FRAP notifies the user.

Evaluation

We set up a Ruby server in a simulated cloud environment. The server handles requests from 10 clients, one of which causes an out-of-memory server crash, a known system level Ruby vulnerability, during both model generation and detection. We experimented with three distance metrics during clustering, Kullback-Leibler, Hellinger, and Euclidean, to compare performance.

Distance Metrics	Isolate at Modeling	Capture at Detection
Kullback-Leibler	Yes	Yes
Hellinger	No	No
Euclidean	Yes	Yes

Table 1: Experimental Results

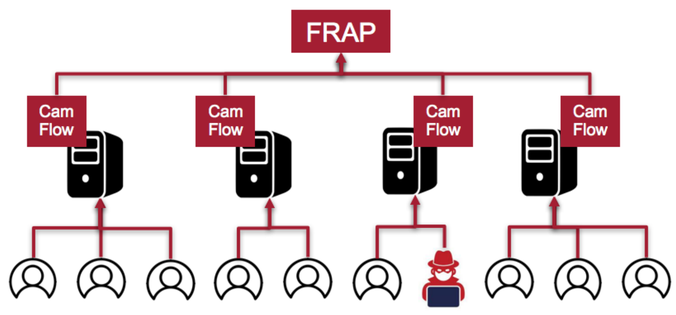

Figure 2: Clients send requests to server instances, whose provenance is captured by CamFlow and sent to FRAP for analysis.

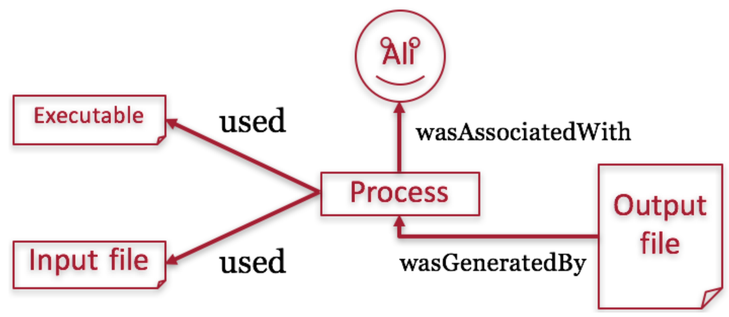

Figure 3: An example of a provenance DAG.

Conclusion

Our early efforts in detecting faults/intrusions through runtime provenance analysis show promise. We will continue to optimize our current prototype and identify machine learning algorithms that will further improve accuracy. Many future research directions unfold. For example, how can we protect system provenance from being tampered with, since trustworthy provenance data is the key to our detection system? How do we provide users with meaningful provenance information to explain the origins of anomalies? We believe current advances in provenance capture systems open a new landscape for research in cloud computing, computer systems, and security.

Download at:
`https://github.com/michael-hahn/frap/tree/master/myapps`

Acknowledgements

This work was supported by the US National Science Foundation under grant SSI-1450277 End-to-End Provenance.

Contact Information

Email: hanx@g.harvard.edu

HARVARD
School of Engineering and Applied Sciences