Process Detection using Hardware Performance Counters

Manish Choudhary *Akshata Rao*

Department of Computer Science
Georgia Institute of Technology
Atlanta, Georgia, USA
{mchoudhary8, arao68}@cc.gatech.edu

Original Proposal

Performance counters can be used to provide useful information about the number of architectural and micro-architectural events such as cache-references, branch-loads and instructions witnessed during the execution of a program. They have been proposed as a method to profile processes to detect anomalous execution of compromised programs, as well as detect malware in [1], [2]. We believe that if process identification is indeed possible through performance counters, it could be applied to the detection of malware that poses as legitimate software and to classify variants of a malicious program in the same category.

Our methodology would involve initially selecting a subset of events that are most suitable to profile a process. This would be used to build a baseline model for a process. The variation amongst different baseline models could be applied to distinguish between processes. We could extend this to further classify malicious and benign behavior. This study would be helpful in understanding whether malicious processes can be detected based on the more reliable hardware features rather than using the software based solutions.

Project Implemented

Our original objective of being able to distinguish between processes by measuring hardware performance counters is unchanged. We began our experiments with selecting the best subset of hardware events based on their values of information gain. This ranked subset was used to build event groups that could be simultaneously measured to collect hardware event counts during the execution of a process. The data collected was used to build a behavioural model for a process, that was further used to distinguish between processes. Clustering algorithms were applied to understand if similar processes were clustered together. In addition, we recorded performance counter data for 2 keyloggers that represented our malicious process set. They were also compared against the dataset of benign processes.

The initial proposal aimed at extending the premise of process identification to actual malware. However, we found that the effort required to establishing a secure yet almost-real environment to run malware successfully without compromising our systems or other machines on the same network, turned out to be beyond the scope of the project. To make up for it, we've performed similar experiments using benign processes, providing enough groundwork to separately focus on malware identification in the future.

Experimental Setup

The data was collected on a VM hosted on VMWare Player on Intel Core i7 processor. The VM was configured to have 4 processors, 1 GB RAM and 20 GB hard disk. The guest operating system was Ubuntu Linux 14.04.

Each event was measured for the purpose of feature selection for an interval of 100 milliseconds for a total duration of 10 seconds. The *perf* tool was run to collect the performance counter values for each group for an interval of 100 milliseconds for a total duration of 120 seconds. The events measured for feature selection purpose can be found in Table 1.

We chose a set of applications varying in activity levels to make the analysis more effective. The processes shortlisted for the study can be found in Table 2.

Methodology

Feature Selection

This task was aimed at finding the most suitable subset of events that provided maximum information about a process. We collected hardware counters data of 26 architectural, micro-architectural and software events for 10 processes. Counters were measured every 100ms for a total duration of 10s. An initialization time of 30-60s was provided for the stabilization of the process. The data collection was repeated after inducing noise by running actively executing process (Chrome web browser rendering a YouTube video).

Three feature selection algorithms: information gain, gain ratio ranking and chi-square ranking were applied to obtain the most useful events that would help identify a process. Our results were cross validated with the set of events obtained from applying the same algorithms on the noise-induced dataset. The 15 best performing events were chosen to form event groups. As it was possible to simultaneously record only 4 performance counter events at a time, we created 14 event groups having 4 events each. The events were combined on the basis of painting an overall picture of a process's execution.

branch-instructions	mem-stores	cache-references	context-switches	iTLB-load-misses
branch-loads	instructions	L1-dcache-load-misses	page-faults	iTLB-loads
branch-load-misses	cycles	L1-icache-load-misses	mem-loads	dTLB-store-misses
branch-misses	stalled-cycles-frontend	L1-dcache-store-misses	LLC-loads, LLC-stores	cache-misses
ref-cycles	dTLB-load-misses	L1-dcache-prefetch-misses	LLC-prefetches	bus-cycles

Table 1 Candidate Events for Feature Selection

Application	**Type**	**Intensity**
OpenOffice	Office Suite	Moderate
Geany	IDE	Light – Moderate
SuperTux	Game	Heavy
Firefox, Chrome	Web Browser	Moderate – Heavy
Foxit	PDF Reader	Light - Moderate
Totem, VLC	Media Players	Moderate – Heavy
7zip	Zip Archiver	High
Logkeys, Linux Keylogger	Keyloggers	Light,Heavy

Table 2 Applications analyzed

Event Groups
L1-dcache-load-misses,bus-cycles,instructions,cache-misses
L1-dcache-load-misses,cycles,branch-misses,cache-references
L1-dcache-load-misses,instructions,cache-references,branch-loads
L1-dcache-load-misses,ref-cycles,instructions,cache-references
branch-instructions,branch-loads,L1-dcache-load-misses,instructions
branch-instructions,mem-stores,cache-references,instructions
branch-loads,branch-instructions,cache-misses,L1-dcache-load-misses
cache-misses,branch-instructions,branch-misses,mem-stores
cycles,L1-icache-load-misses,L1-dcache-prefetch-misses,branch-loads
cycles,L1-icache-load-misses,branch-load-misses,cache-references
instructions,L1-dcache-store-misses,branch-misses,cycles
instructions,branch-loads,L1-dcache-store-misses,branch-instructions
instructions,branch-misses,mem-stores,cache-references
mem-stores,instructions,cache-misses,cache-references

Table 3 Event groups formed from the top ranked events

Clustering using Unsupervised Learning

We used self-organizing maps, a clustering algorithm, from the WEKA suite[3] to help us identify similarities and differences between execution behaviours of multiple processes. Event data was collected simultaneously for the 4 listed events in each group, for a total duration of 2 minutes, with sampling intervals of 100 ms. Around 1200 feature vectors were collected for each of the 14 groups for all the processes under study.

Behavioural Modelling using Supervised and Unsupervised Learning

Data was collected similar to the methodology described for clustering. Self-organizing maps, an unsupervised classification technique, was used to build a baseline behavioural model for a set of 8 processes. Variation in process behaviours were induced through parametric changes and inducing noise through a second active process running in the background.

Experimental Results

Experiment I : Clustering

The aim of the experiment was to get a basic understanding whether performance counters were a feasible option to distinguish between different processes.

Results and Analysis

The results [Figure 1] show that processes with different behavior fall in different clusters while processes with somewhat similar behavior are clustered together. SuperTux and 7Zip are both very heavy processes and thus, get clustered together. Similarly, Firefox and Chrome are web browsers and thus, are expected to show similar behavior for same kind of load. The results show that they both get clustered together. Finally, the remaining processes are grouped in one cluster, their execution behaviors being moderately heavy. Therefore, we can conclude that performance counters can be a good option for categorizing similarly behaving processes.

Experiment 2 - Behavioral Modelling Using Unsupervised Learning

This experiment was performed to understand if we can exploit the hardware performance counters to model the behavior of processes. This behavioral model would be used to classify/categorize these

processes. The same data collected for clustering was used in this experiment. “Self Organizing Map”, an unsupervised classification technique, was implemented in Python to identify if the learning and training sets get classified as a single process or separate ones.

The below analysis was performed:

- What does happen when different processes are compared?
- What does happen when variants (different inputs/loads) of the same process are compared?
- What is the impact of noise on the process identification accuracy?

We compare the classification accuracy across all the 14 groups to shortlist the three best and the three worst performing groups [Figure 2]. The best performing groups have 60-70% accuracy while the worst groups are able to predict with 40-50% accuracy.

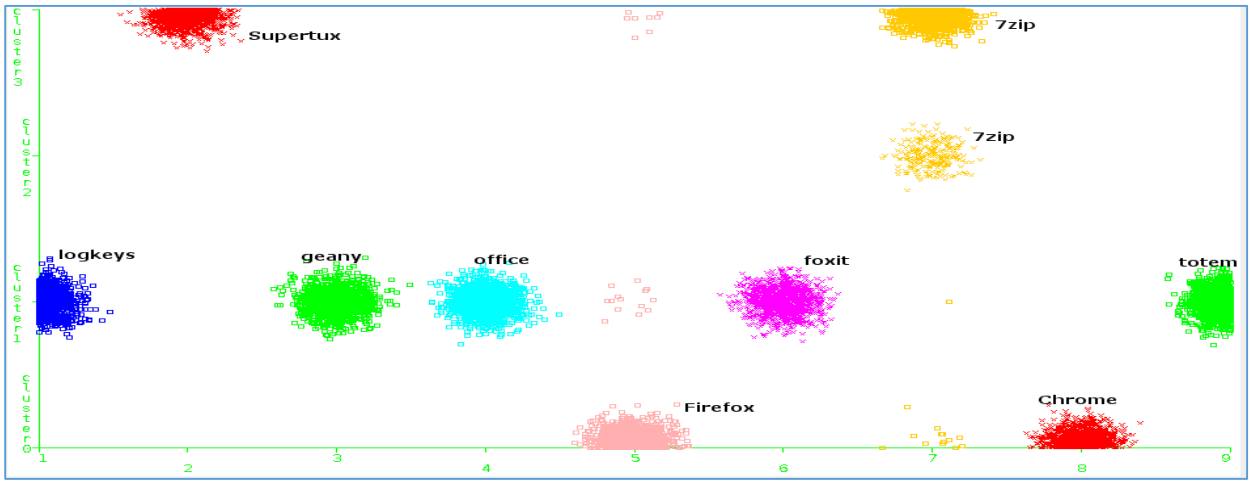

Figure 1 Clustered Processes (X-axis – Classes, Y-axis – Clusters)

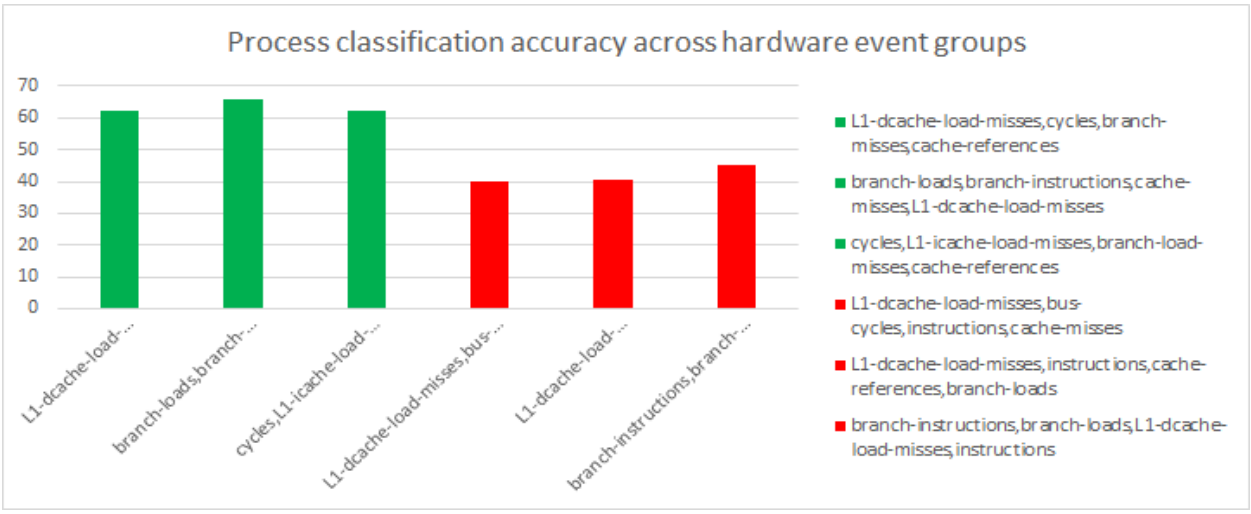

Figure 2 Process Classification Accuracy for the best and worst 3 performing event groups

However, the best performing groups are not constant across all processes. Thus, making it necessary to select a new set of best performing groups per process to achieve the possible classification accuracy.

Additionally, no event emerged as the single most useful event for identifying processes. Hence, we recommend that events in a group be chosen to cover different architectural / micro-architectural aspects.

Inter-Process Distinction - Analysis

We attempt to distinguish between two different processes using the behavioral models built using the 14 groups. We found that three groups, on an overall basis, were comparatively most successful in discerning between processes. However, per process, the set of the best 3 performing groups varies.

The plots below display the detection accuracies for a subset of the processes for the three overall best groups and the three best groups specific to each process.

The behavioural model based on Firefox [Figure 3] is successfully able to distinguish between Firefox and most other processes, with the process specific best three groups have a higher classification accuracy. The same may be said for Geany[Figure 4] and Totem[Figure 5].

This suggests fixing an event group across processes may not be practical for process detection. We also noticed for certain processes, even the best performing groups may not have the highest classification accuracy.

The plots display the accuracy of distinguishing between processes for the event groups that performed best overall across processes, and for the specific process being studied.

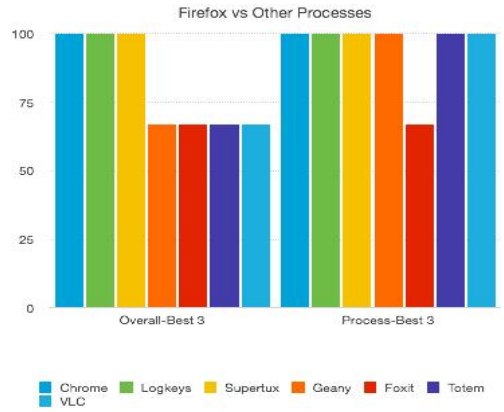

Figure 3 Accuracy of distinguishing between Firefox and other processes

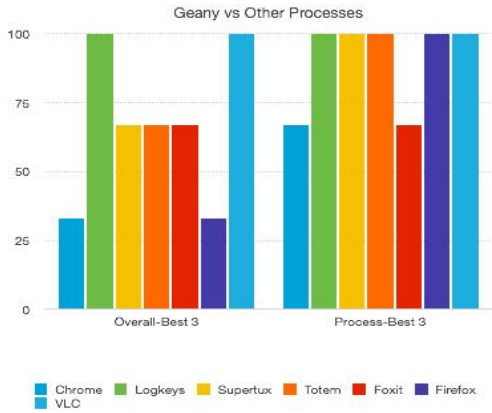

Figure 4 Accuracy of distinguishing between Geany and other processes .

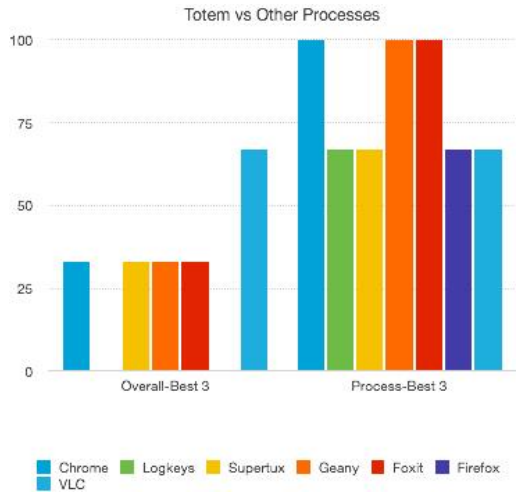

Figure 5 Accuracy of distinguishing between Totem and other processes

Intra-Process Identification - Analysis

We attempt to find the detection accuracy for a process in the presence of variations in inputs or loads. Once again, a baseline model was built on the vanilla run of the process and compared with the data collected on process variations. Results for Totem [Figure 6] were generated by running MP3, HD Video and MP4 files. The results shown correspond to the three groups that were most accurate in identifying Totem. The prediction accuracy is high for different types of inputs except when we used a merged data set for training Totem's behavioral model. The combined dataset comprises of datasets that represent different behaviors of Totem (e.g. with MP3, HDVideo etc). This result showed that building a generic model for a process by training on varying inputs may not always improve accuracy. Process detection accuracy results of the overall best three groups and process specific best three groups have been shown for Firefox [Figure 7], 7Zip and Open Office [Figure 8]. These results also conclude that process specific groups are better than universally used single group to model processes.

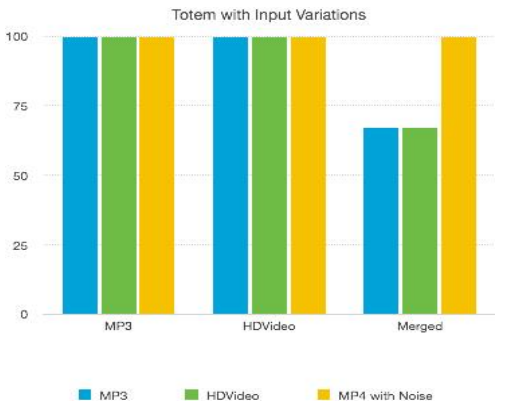

Figure 6 Accuracy of identifying Totem with input variatiions

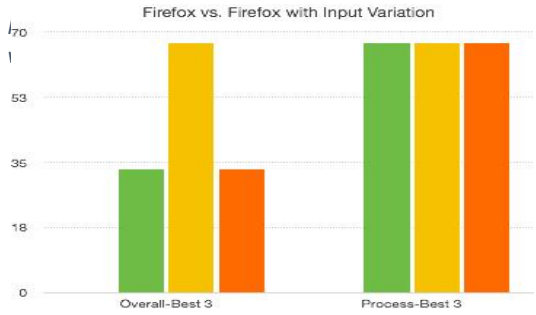

Figure 7 Accuracy of identifying Firefox with input variations

Noise Effects on Identification - Analysis

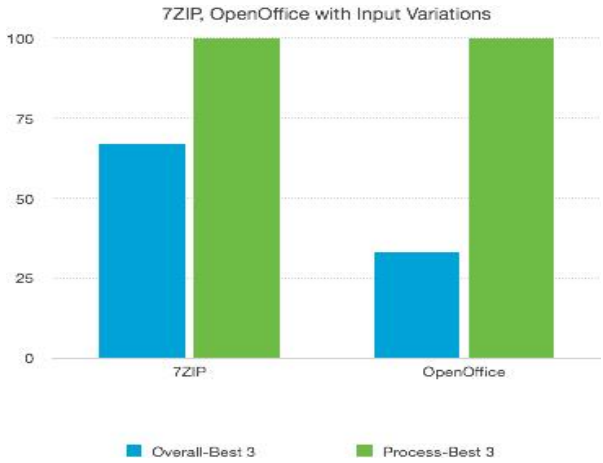

Figure 8 Accuracy of identifying 7zip, OpenOffice with input variations

The impact of noise on the process prediction accuracy was also studied. [Figure 9] shows some interesting results.A YouTube video was rendered on Google Chrome web browser to create a noisy environment. We performed two experiments. We trained our algorithms on datasets that were collected in absence of any noise (Vanilla Learning) and used it to predict the process for datasets collect in the presence of noise. The reverse (Noisy Learning) was also executed.

The results suggest that the impact of noise is negligible if the observed process is heavier than the process producing the noise. "Totem playing HD video" and 7Zip are heavier than Chrome and thus, we are able to predict with high accuracy even in presence of noise. But, as the modelled processes become comparable or lighter than the noisy process, the noise has increasing impact. This can be concluded by observing that the prediction accuracy of the groups is not good for Geany, Totem-MP3 and OpenOffice which are, in comparison, equivalent to or lighter than Chrome. The plots display the variation of process identification accuracy across varied behaviors of the process.

Experiment 3 - Classification of Known Processes

This experiment tries to answer the following question - *"If we have a pre-determined set of processes, how feasible is it to correctly classify newly*

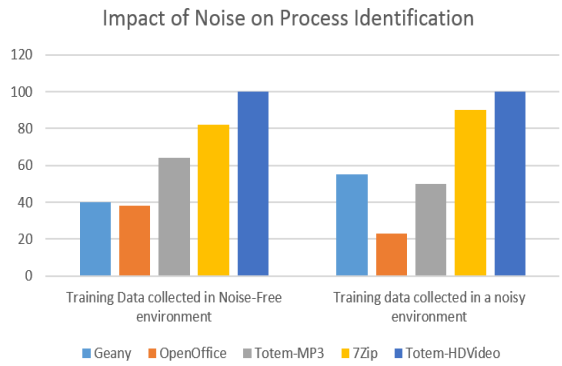

Figure 4 Impact of noise on process identification

collected data for a process in this set?". We built our learning set with multiple data sets of vanilla runs (without noise and limited input variations) of the different processes. Decision Trees classification algorithm from the WEKA[3] suite was used to predict the process.

We found that the classification accuracy is high for certain event groups on a test data set which included data collected both through input variations to the processes and a noisy environment. The plot below shows the variation of the classification accuracy across event groups.

Observations

We observed that in a stable environment, it's possible to distinguish between two processes whose operation differs widely. However, for similarly behaving processes, as seen in the case of Totem-VLC and Firefox-Chrome process pairs, it is much harder to distinguish between them. Additionally, if a process itself has widely varying behavior depending on parametric inputs or noise, it may be necessary to treat each behavior of the process as a separate entity to learn on. If the unknown process is known to belong to a fixed set of previously studied processes, then the probability of correctly classifying the process according to its family is high. On a side note, we also encountered processes such as Firefox, which did not fit into a particular behavior model showing incomprehensive clustering / classification patterns in its data.

Conclusion

The takeaway was that hardware performance counters gave you some insight into the identity of the process. However, rather than building a one-size-fits-all solution, we need a tailored dataset for the processes we intend to learn on, to achieve higher data accuracy. In conclusion, there is some scope for extending our research to applications such as malware detection and identification, as well as protection against the process vulnerability exploitation and control flow hijacking.

References

[1] Unsupervised Anomaly-based Malware Detection using Hardware Features - *Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo*

[2] On the feasibility of online malware detection with performance counters. - *John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo* - In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (ISCA '13). ACM, New York, NY, USA

[3] WEKA Suite [http://www.cs.waikato.ac.nz/ml/weka/]

[4] Perf Tool [https://perf.wiki.kernel.org/index.php/Main_Page]

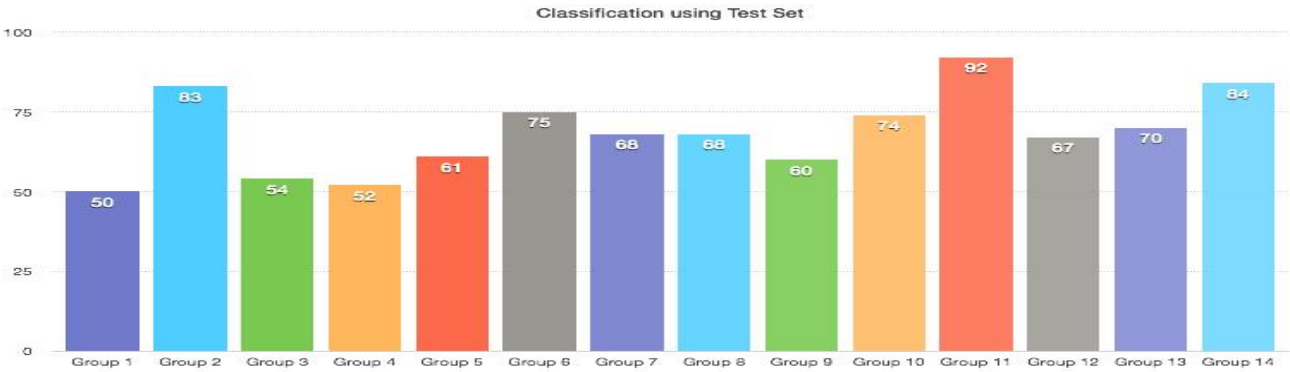

Figure 10 Classification Accuracy of Processes with split data set