**AFRL-IF-RS-TR-2005-278**
**Final Technical Report**
**July 2005**


# APPLICATION-LEVEL ANOMALY DETECTION FOR THE MASTER CAUTION PANEL

**Columbia University**

**Sponsored by**
**Defense Advanced Research Projects Agency**
**DARPA Order No. P093**



**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**
**ROME RESEARCH SITE**
**ROME, NEW YORK**

**STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-278 has been reviewed and is approved for publication

APPROVED: /s/

GLEN E. BAHR
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | JULY 2005 | Final Sep 02 – Jun 04 |

**4. TITLE AND SUBTITLE**
APPLICATION-LEVEL ANOMALY DETECTION FOR THE MASTER CAUTION PANEL

**5. FUNDING NUMBERS**
C - F30602-02-2-0209
PE - 62301E
PR - P093
TA - CP
WU - 04

**6. AUTHOR(S)**
Salvatore J. Stolfo

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Columbia University
Trustees of Columbia University in the City of New York
Office of Projects and Grants
1210 Amsterdam Avenue, Mail Code 2205, 254 Engineering Terrace
New York New York 10027

**8. PERFORMING ORGANIZATION REPORT NUMBER**
N/A

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGB
525 Brooks Road
Rome New York 13441-4505

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
AFRL-IF-RS-TR-2005-278

**11. SUPPLEMENTARY NOTES**
AFRL Project Engineer: Glen E. Bahr/IFGB/(315) 330-3515/ Glen.Bahr@rl.af.mil

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** ***(Maximum 200 Words)***
The goal of this work was to study how to monitor a large distributed system and apply machine learning methods to, and generate models of, its normal operation. With this done, the generated model(s) may be used to actively detect abnormal executions at run-time which may indicate improper use, attacks, or internal faults of the monitored system in question. We used the data collected by Master Caution Panel (MCP) software for the Theater Battle Management Core System (TBMCS) as sample data to test our machine learning methods. The MCP system has been under development for some time. It shares the same goal, but is based upon carefully designed and crafted "logic modules" that issue alerts when conditions warrant. The goal here is to use the existing monitoring and alert functions of MCP as a baseline to determine whether automated learning systems can achieve comparable performance in an automated fashion. A positive outcome of this study could suggest general principles of use in a wide range of mission critical systems.

**14. SUBJECT TERMS**
Cyber Panel Program, Master Caution Panel, Distributed System, Anomaly Detection, Machine Learning, Probabilistic Anomaly Detection Algorithm

**15. NUMBER OF PAGES**
17

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

# Table of Contents

# List of Figures

# 1. OVERVIEW

The goal of this work was to study how to monitor a large distributed system and apply machine learning methods to, and generate models of, its normal operation. With this done, the generated model(s) may be used to actively detect abnormal executions at run-time which may indicate improper use, attacks, or internal faults of the monitored system in question. We used the data collected by Master Caution Panel (MCP) software for the Theater Battle Management Core System (TBMCS) as sample data to test our machine learning methods. The MCP system has been under development for some time. It shares the same goal, but is based upon carefully designed and crafted "logic modules" that issue alerts when conditions warrant. The goal here is to use the existing monitoring and alert functions of MCP as a baseline to determine whether automated learning systems can achieve comparable performance in an automated fashion. A positive outcome of this study could suggest general principles of use in a wide range of mission critical systems.

Section 2 describes how the current MCP software monitors the system and the logic used to generate alerts. Section 3 gives a brief description about the Probabilistic Anomaly Detection (PAD) algorithm we used in this study to generate alerts. PAD was designed and implemented in our previous research supported by the DARPA CyberPanel program. Section 4 details our test results using the phase1, 2, 3 data from SRA Adroit Center MCP Lab, and compares the alerts generated by the PAD to those generated by MCP. We thus have a means of determining how well PAD modeled MCP's behavior, and how well we conformed to the hand-crafted rules of MCP. The phase 1 dataset was gathered from MCP and consists of a base set of data to reveal the necessary detail on sensor data formats. Phase 2 consisted of a set of data with purposefully injected faults to determine MCP's alarm behavior used as a baseline; while Phase 3 was a similar dataset, but included a more substantial monitoring period. Considerable effort was involved in debugging MCP monitor data and elucidating some detailed aspects that were not immediately apparent in the preliminary documents received. The final section describes subsequent work performed on applying the anomaly detection algorithms to content data, as well as file system accesses on the various hosts implementing core elements of a distributed application.

# 2. MCP

The Master Caution Panel (MCP) software monitors the Theater Battle Management Core System (TBMCS) and analyzes the collected data to generate alerts. MCP is organized as a three-level monitoring system: The lowest level captures audit data from deployed sensors at various sites in the distributed system; the raw audit data is called RAWEVENT. The middle level of MCP processes the RAWEVENT data stream sent from the lowest level sensors to generate SUBEVENTs that essentially aggregate information into more meaningful state information about the operation of TBMCS. The highest level checks the SUBEVENTs (or combinations thereof) to generate ALERTEVENTs. An ALERTEVENT with a "DOWN" value means there is something wrong with the TBMCS system.

The RAWEVENT stream is collected in a periodic fashion (every 20 seconds) by JavaScripts dispatched to each host computer, and is then sent to upper processing

levels for analysis. We applied PAD to this data stream, working directly on the RAWEVENT data to determine whether we may generate the same alerts as the analytics embedded in MCP at the higher levels.

MCP collects 12 kinds of RAWEVENTs that describe the process state of each host on which a sensor is placed. They are: **cpuUsage, totalSwap, reservedSwap, availSwap, allocatedSwap, swap, pingCollection, unixProcessList, activeProcessInfoVector, activeProcessDbSet, databaseOk.** Among them, **swap** actually contains all the information of **totalSwap, reservedSwap, availSwap, allocatedSwap.**

There are only three kinds of SUBEVENT data generated by MCP. They are: **unixProcessInfoVector, pingFailureVector,** and **activeProcessInfoVector.** They are generated from RAWEVENT data: **unixProcessList, pingCollection** and **activeProcessInfoVector,** respectively. The RAWEVENT **unixProcessList** is the information about all the processes running on each machine at the time of data collection. The sensor in this case is simply the Unix "ps" command. The RAWEVENT **activeProcessInfoVector** contains the active processes accessing the underlying TBMCS database from all the hosts at during the collection period. The RAWEVENT **pingCollection** describes the status of each host (alive or dead) and its ping time.

There are two kinds of ALERTEVENTs. One is triggered by the **pingFailureVector** data and the other is triggered by a combination of **unixProcessInfoVector** and **activeProcessInfoVector** data.

Although MCP collects 12 kinds of RAWEVENT data, the alerts generated in the data sets provided in this study apparently only used 3 of these events to generate alerts. (This may mean that only a subset of possible failure states of TBMCS have been exercised for this study.) For **unixProcessList** and **activeProcessInfoVector,** MCP analyzes only three specific processes: **aps_assess, apsqtp** and **apsui**. These three types of process are database-related. If there is a process running on some host, the database server should also have information about the process.

The logic to generate an alert in the current MCP implementation appears to be as follows:

1. When there is some host that has changed state from alive to dead; the ping sensor data changes from live to dead implying that the host has crashed, or is hung, its network access is disconnected.

2. Among current **unixProcessInfoVector** and **activeProcessInfoVector** SUBEVENTs that are received, if there is some process (one of the **aps_assess, apsqtp** and **apsui** processes) from **unixProcessInfoVector**, but is not in the **activeProcessInfoVector**; this condition generates a "DOWN" alert. This situation means there is an orphaned process with no information in the database.

The analysis we performed using the probabilistic anomaly detection algorithm is applied to the same data sources. We first wish to determine whether the same conditions that generate alerts in MCP will manifest as low probability or anomalous data as detected by PAD. Further, we wish to determine whether there are other low probability data events that might also generate alerts. Given the short duration of data

acquisition, it is not possible to provide an adequate assessment of the latter case without further injecting additional faults and running the application for a far longer period of time. Training a PAD model during a realistic application of TBMCS in an AOC-type environment (or in a user training environment) would be a minimum ideal set of data to analyze.

# 3. PAD ALGORITHM

The Probabilistic Anomaly Detection (PAD) algorithm is an efficient and robust algorithm that can process a huge amount of data in real time. It has been designed for data consisting of a set of features, each with a large number of discrete values. The basic idea behind PAD is that it trains over a data set and then checks to see if newly observed data are "consistent" with the previous ones.

In general, a principled probabilistic approach to anomaly detection can be reduced to density estimation. It estimates a density function $p(x)$ over the normal data, and defines anomalies as data elements that occur with low probability. But in practice, estimating density is a very hard problem, especially for sparse data. Instead, PAD defines a set of consistency checks over training data, and any data that fails one of the consistency checks will be treated as anomalous.

Basically there are two kinds of consistency checks: first order and second. First order consistency checks evaluate whether a single feature is consistent with other features in the dataset. Second order consistency checks evaluate if a pair of features is consistent with the data set. All consistency checks are evaluated by computing a predictive probability. This is done by estimating a multinomial using counts observed from the data set and then estimating the probability of seeing the observation. One of the clever aspects of PAD is the manner in which it estimates the probability of seeing unobserved data.

Suppose each data record has $n$ features; we treat each feature as a random variable and write the data record as a set of random variables $x_1, x_2, ..., x_n$. Now each first order consistency check produces a score by computing $p(x_i)$, which is the likelihood of an observation of a given value of feature $x_i$. Second order consistency check is denoted by $p(x_i \mid x_j)$, which determines the conditional probability of a feature value given another feature value, where $x_i$ and $x_j$ are the feature variables.

To estimate the probabilities we use the estimator presented in [Friedman and Singer, 1999], that explicitly estimates the probability of observing a previously unobserved element. The details about the estimation formula will be skipped here. Please refer to the paper "Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses," Stolfo, Apap, Eskin, Heller, Hershkop, Honig, and Svore, *CU Tech Report* Feb. 23, 2004. The essence of the idea is to estimate the likelihood of seeing new distinct values of a feature given the range of possible values the feature may take on and the proportion of distinct values observed while training.

By way of example, consider a feature of a TCP packet data stream, **the source IP address**. The range of values of a source IP may span the entire 32-bit address space. But in a particular data stream one may observe, what is the likelihood of seeing a

distinct new IP address that has not been previously observed during some training period? Consider two cases, peering points and enclaves. Within a peering point, it is very likely that many distinct IP addresses will be observed simply since the peering point serves internet connectivity for a broad range of the network. For any period of time, it would be highly likely a new IP address will therefore be observed. In an enclave LAN environment, where there are a fixed number of assigned IP addresses, it is likely that during a sufficiently long training period all IP addresses within the enclave will be observed. The likelihood of seeing a distinct IP address not observed during training in this context would be highly unlikely. This principle is applied in PAD to produce a more robust estimator of the probability density of the training data.

PAD is an unsupervised learning algorithm. One simply provides a set of discrete valued feature vectors to the algorithm, and a model is computed that consists of a number of "consistency checks". These checks score a data item using the estimated probability density function as being consistent with the observed training data, or not. The decision process is based upon a threshold applied to the computed scores.

In the experimental setting for this project, we input data supplied by the MCP sensors, apply PAD, and output a model. The model is then used to score the original data to find those data that are inconsistent with the rest of the training data. There are NO labels associated with the data. The core thesis is that data found to be inconsistent, low probability events, will have been generated by states of the TBMCS system associated with internal failures or faults, or misuses of the system. If this were not the case, then TBMCS would experience faults frequently, which we believe to be counter-intuitive. Further, PAD may detect certain low probability events that are not truly associated with a faulty state of TBMCS, but may instead be rare states exhibited by the system. This may be due to a lack of a sufficiently long period of time of observing the system in operation.

In summary, we applied the PAD algorithm to only the RAWEVENT data of MCP data, which is the lowest level data gathered from the operation of TBMCS. The PAD models so generated should be a faithful representation of operation in practice of TBMCS. In principle, any "unlikely" data detected from the RAWEVENT stream ought to indicate a very unusual state of TBMCS, which may occur due to a fault, a misuse of the system, or some non-harmful but unusual state not observed previously. The determination of which of these three cases may be the cause for the anomalous data requires further drill-down and study.

# 4. EVALUATION

## *4.1 Numerical Number Preprocess*

PAD is originally designed for the features with discrete values, but in MCP there are several RAWEVENT data with numerical values. For example, **cpuUsage** and **swap** each are measured as numerical quantities. To apply PAD to these RAWEVENT data, we need to preprocess and discretize the data, otherwise the false positive will be too high. For example, **cpuUsage** 7.8999 and 7.9 should be treated as same value. If we use the numerical number directly, 7.8999 might appear only once or twice among all the data and will be caught as alert by PAD, which will cause false positive.

To avoid such problems, we approximate the number using a Gaussian distribution. We treat the numbers as a sample to compute the average and standard deviation, and then bin the data accordingly. The width of each bin equals the standard deviation, and the average is the center of the bin. The values falling in a bin will all be represented using the centroid of that bin. For example, if the average is 20 and the standard deviation is 4, then the bins will be [10,14), [14,18), [18,22), [22,26),… The value 13.2 will be represented by 12, and 22.1 will be represented as 24.

The width of the bin can be adjusted, but the idea behind this is to generate data-dependent discretization of observed numerical values. Because PAD detects the low-probability events, using Gaussian approximation is a reasonable choice. We can study more accurate regression methods later, for example, radial basis functions.

## *4.2 RAWEVENT Representation*

We experimented with two different representations of the RAWEVENT data, which reflect two different aspects about the system status. The first representation used is "native," which is the RAWEVENT directly collected from MCP; the second representation is the "dynamics" of the audit stream; that is, we computed features that represent the information that changed in the system over consecutive checkpoints. The native representation is referred to as "Approach I" in the performance graphs displayed in the performance section of this report, while Approach II refers to the derivative features extracted from the data.

For example, when considering the RAWEVENT **cpuUsage**, we observe a series of numbers collected from the system periodically, e.g., 13.5, 84,7, … ,86.3, 99.4, and so forth. Using the first approach, we can use these numbers directly and discretize them as described above. These feature values are then used directly in PAD. If the average of **cpuUsage** is 30 and standard deviation is 15, the **cpuUsage** of 99.4 might trigger an alert because it means the current system has too heavy a computational load.

However, in the second representation, we consider the amount of change in these numerical values. For these derivative numerical features, for example, the numbers computed from the sequence of raw data collected become: 71.2, … , -10.3, 13.1, and so forth. If the average of these numbers measuring the change in **cpuUsage** is 3.42 and the standard deviation is 4.58, the observed value of 71.2 might trigger an alert because that means the system had a sudden abnormal huge increase in computational load. The reason for such a sudden huge change might indicate a fault in the system. Or, as determined by training in PAD, it may not be an unusual event.

For the **unixProcessList** and **activeProcessInfoVector** data of the RAWEVENT stream with discrete values, we treat them as a set, and the change of status is the difference of the two consecutive sets $(A-B)\cup(B-A)$, with their difference in size $|B|-|A|$, (given that $B$ occurs after $A$).

Given these two representations of the MCP data stream, we next describe the results of our automated training and analysis to determine whether or not we may generate a model consistent with current MCP logic rules that generate alerts.

### 4.3 Data and Model Details

We will use **pingCollection** RAWEVENT as a simple example to illustrate the work of PAD. A piece of the phase3 raw data is as follows:

```
<RAWEVENT id="RAWEVENT.mcpsv1.ping.pingCollection.345"
        hostname="mcpsv1" datetime="06.26.2003 16:50:09:407 BST">
<PROPERTYCHANGED name="pingCollection" scriptName="ping">
<NEWVALUE><PING_COLLECTION>
<PING targetHost="128.132.41.67" result="false" latency="2147483647"/>
<PING targetHost="128.132.41.71" result="true" latency="10"/>
<PING targetHost="128.132.41.73" result="true" latency="10"/>
<PING targetHost="128.132.41.78" result="true" latency="10"/>
<PING targetHost="128.132.41.69" result="true" latency="10"/>
<PING targetHost="128.132.41.64" result="true" latency="10"/>
<PING targetHost="128.132.41.70" result="true" latency="10"/>
<PING targetHost="128.132.41.74" result="true" latency="10"/>
<PING targetHost="128.132.41.77" result="true" latency="10"/>
<PING targetHost="128.132.41.56" result="true" latency="10"/>
</PING_COLLECTION></NEWVALUE>
</PROPERTYCHANGED>
</RAWEVENT>
```

Because the targetHost are fixed, we will use only result and latency as the features for **pingCollection** RAWEVENT. So the input to PAD (as a line delimited set of feature values) will be:

```
… …
false 2147483647
true 10
true 10
… …
```

The formatted data file size is 182760 bytes, and the model generated by PAD for the raw data is only 4666 bytes.

The result after running PAD appears as follows (here with two feature values, followed by 6 scores):

```
false 2147483647 : -4.744362 -4.460786 0.676618 0.256720 0.536305 0.680725
true 10 : 0.688787 0.687354 0.693076 0.692501 0.691068 0.693075
```

There are 6 possible scores that may be generated for each record containing two features. If we denote the first column as $x_1$ and second column as $x_2$, the six consistency check scores computed are for $p(x_1), p(x_2)$, $p(x_1 \mid x_1), p(x_1 \mid x_2)$, $p(x_2 \mid x_1), p(x_2 \mid x_2)$, respectively. We don't use $p(x_1 \mid x_1)$ and $p(x_2 \mid x_2)$, since they are redundant. Because of numerical computation issues, (e.g., possible underflow,) the values are the log of the probability estimate. The smaller the value, the lower the probability the event is consistent with the training data. So those records with score values smaller than some threshold will be treated as abnormal events. The threshold can be adjusted to control the hit rate and false positive rate. If we set the threshold to

-4 for the above example, the record “false 2147483647” will be deemed “abnormal” and trigger the alert; while the record “true 10” is deemed normal.

### *4.4 Performance Comparison*

Since the MCP ground truth in the datasets provided is limited to three RAWEVENTs that are used to generate two kinds of alerts, we first compare how well PAD can compare MCP alert output for these three events. We also generated alert output for all data and describe these results later in the discussion section. Since there is no ground truth for these cases, we can only speculate about the effectiveness of the method.

## 4.4.1 PingCollection

There are 10 hosts in the whole system, so each **pingCollection** RAWEVENT contains the ping time information about all these 10 hosts. In this experiment we use the native MCP audit data with discretized numerical features. Applying PAD to the **pingCollection,** we get:

| RAWEVENT | Alert of MCP | Alert of PAD | Hit rate | False Positive |
|---|---|---|---|---|
| pingCollection | 10 | 29 | 100% | 4.2% |

From the table we can see PAD can successfully catch all those alerts that MCP generated but with a false positive rate of 4.2%. But upon inspection of the data deemed anomalous by PAD, we found the 19 “extra” alerts from PAD are indeed true alerts, not false positives. Each of the RAWEVENTs that generated the 19 alerts reveals that one or more hosts are unreachable. The reason why MCP only produces 10 alerts is that MCP includes an “alarm suppression” feature that will not generate a continuous stream of alerts for the same host. If an alert is generated by MCP that indicates a host is unreachable, in the following sensor data collected in the RAWEVENT stream, MCP won’t issue more alerts. No such alert suppression scheme is presently implemented in PAD, so it correctly generates alerts for these conditions. This means PAD can successfully detect all the alerts accurately about unreachable hosts with a **zero** false positive rate.

## 4.4.2 unixProcessList and activeProcessInfoVector

MCP uses the combination of SUBEVENTs **unixProcessInfoVector** and **activeProcessInfoVector** to generate alerts. This requires time synchronization. SUBEVENTs occurring at the same time might be triggered by RAWEVENTs gathered at slightly different times. It’s particularly difficult to determine the exact pair of **unixProcessList** and **activeProcessInfoVector** RAWEVENTs used by the logic modules of MCP and the means employed to synchronize this information. So we use **unixProcessList** and **activeProcessInfoVector** separately. As we said before, because MCP only considers three types of processes, we extract only the information about those three types of processes and compare the result to the case where all the processes are included in the model.

We have the following results:

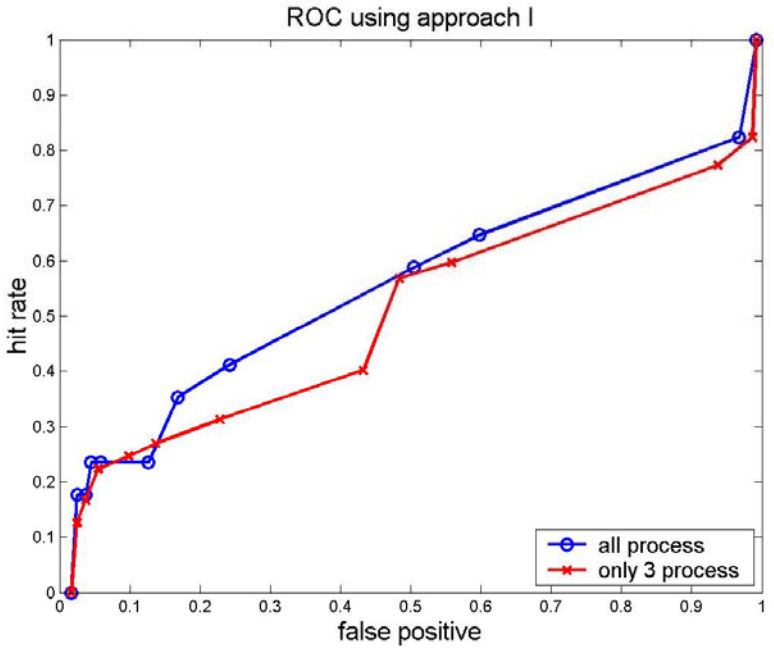


**Figure 1. ROC curve for activeProcessInfoVector RAWEVENT when using the "native" RAWEVENT information. The plot labeled "only 3 processes" is the case when we model only the three types of processes: aps_assess, apsqtp** and **apsui.**

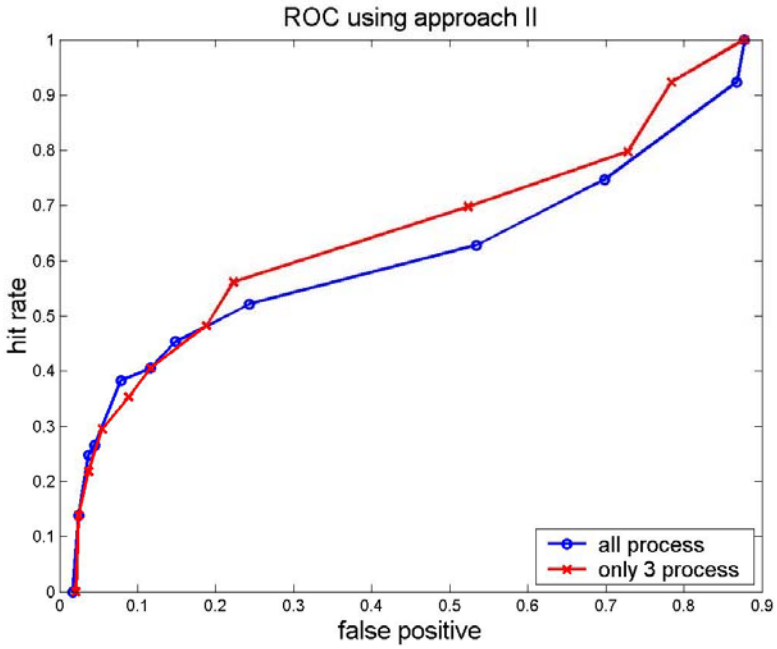


**Figure 2. ROC curve for activeProcessInfoVector RAWEVENT when using the "Dynamic" RAWEVENT information. The plot labeled "only 3 processes" is the case when we model only the three types of processes: aps_assess, apsqtp and apsui**

From Figures 1 and 2 for **activeProcessInfoVector** RAWEVENTs, we can see that considering only the 3 types of processes won't help. That's because these three types of processes are common processes and appear often. It is not abnormal if they exist. On the contrary, there might be some problem with the system if they disappear from the RAWEVENT data. That's why using the second approach, the representation that considers the dynamics of the system's operation, produces a better result. But all these results are not satisfactory.

From the above analysis we think filtering out only those three types of processes cannot help increase the performance. For the **unixProcessList** RAWEVENT, we use the information about all the processes and compare our results between the two representations, "native" and "dynamic".

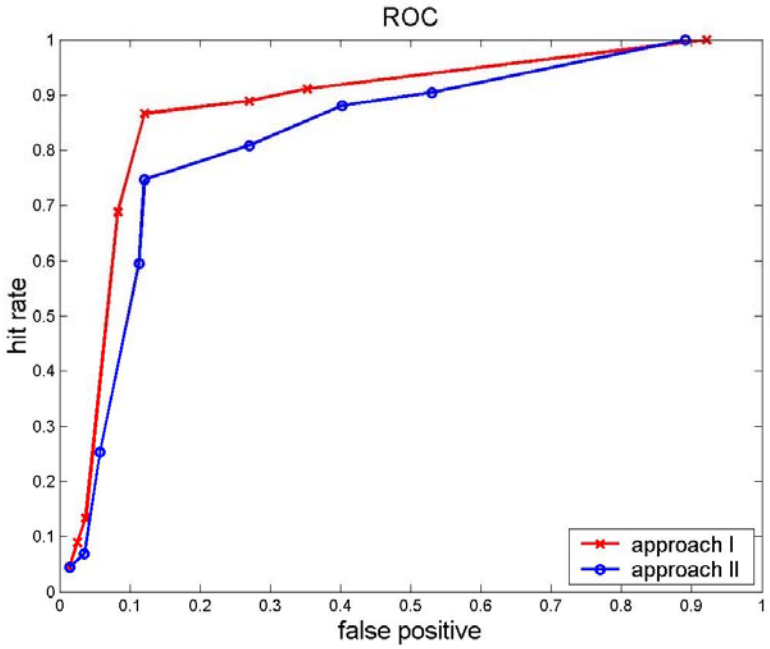


**Figure 3. The ROC curve of unixProcessList RAWEVENT using information about all of the processes. The comparison is between representation approach I and approach II.**

Comparing Figure 3 with Figures 1 and 2, we see that the models employing the **unixProcessList** RAWEVENT data produces a better result than **activeProcessInfoVector** RAWEVENTs. This is perhaps because **unixProcessList** gives more information about the system, while **activeProcessInfoVector** is only database related. Combining them might improve the final result if we can find a good way to pair these two sources of information together in one model.

### 4.4.3 swap and cpuUsage

MCP didn't use the information of **swap** and **cpuUsage** to generate alerts, so we cannot produce a hit rate and false positive comparison. But PAD does find some abnormal events that happened very infrequently. For example, in the phase 2 data set, the host **opcli01** had **cpuUsage** 97.0 and 96.5, while the average of **cpuUsage** is only

9.23 with a standard deviation of 9.43. But for host **plcli02**, **cpuUsage** of 97.0 is pretty normal, since its average is 90.7 with a standard deviation of 25.43. PAD can successfully generate alerts of **opcli01** and didn't introduce false alerts to **plcli02**. Similar results apply to **swap** RAWEVENT data.

The TBMCS system might work well under these situations of measured cpu usage and swap space, but perhaps issuing alerts to cases where sensors measure very high CPU usage or very low available swap space may be a good indicator of the system's tenuous state. At least we know this state is not usual in TBCMS operation.

### *4.5 Discussion*

The experimental result shows that machine-learning method can work effectively on MCP data on some kinds of RAWEVENTs, but does not work very well on other events. We are not sure about the reasons and need more exploration of the data. Intuitively, the following aspects might have affected the results:

1. First, there might not be enough data to train an effective model. We have only three datasets, each around 150 MBytes. The phase1 dataset was configured wrong , so we didn't use it. Phase 3 also seems to have a configuration error. Each of the data sets is about a 4-hour run of TBMCS, which may be far from enough time to reveal sufficient span of the operational states of the system.
2. The logic to generate alerts in MCP is very restricted, and its use of the combination of SUBEVENTs that generate alerts makes it very hard to catch the exact same alerts from RAWEVENTs directly. This partially explains the bad ROC curves.
3. There might be some internal error in MCP logic. Using SUBEVENTs instead of RAWEVENTs introduces some delay, especially with a 20-second cycle time of sensor data acquisition; and it is hard to synchronize the whole system in its current architecture. Some sensor data is gathered on a different time scale, and correlating the separate data sources is not possible. Thus, the alerts may be generated from RAWEVENTs at different points in time which may not reflect the status of the whole system in a consistent way. It is possible that this monitoring architecture could have missed some real alerts and generated some false alerts. MCP continues to be developed, and we suggest that effort be devoted to establishing a common cycle time for all sensor data acquisition.

## 5. Subsequent Research Work

The work stated above is just an initial step towards the goal of applying machine learning methods to detect anomalies in application level data. There are several important areas of investigation we pursued as next steps of our work:

1. Improve the PAD algorithm to:
    a. handle numerical feature values directly as a mixture of Gaussians.
    b. add the capability to do feature selection automatically
    c. provide the means of "pruning" features from the model that can't possibly generate anomalous scores and

d. provide a means of calibrating PAD score threshold in an automatic fashion.

2. Consider applying other related machine learning methods. Modeling the dynamics of sensor data using Hidden Markov Models might be better, especially when combined with PAD models.
3. MCP is a convenient study case for this work and it would be worthwhile to continue working collaboratively with Rome Labs and the MCP team. We hope to develop a general framework that can be applied to any large distributed system; and this may require a deeper understanding of exactly what sensors would work best in a general setting. The current set of sensors is obviously driven by available tools in Unix (ps, ping, etc.).
4. We believe it was necessary to perform a deeper analysis involving the payload data exchanged between processing hosts within the system might reveal other important information of value in detecting faulty states of the system. Payload analysis is an open topic for research. We briefly describe our results next.

# 6. Payload Anomaly Detection

We conceived of a novel payload-based anomaly detector, we call PAYL, for intrusion detection. PAYL models the normal application payload of network traffic in a fully automatic, unsupervised fashion. The method we choose is very efficient; our goal was to deploy the detector in high bandwidth environments either on a firewall, a network appliance, a proxy server, or on the target hosts. We first compute during a training phase a profile byte frequency distribution and their standard deviation of the application payload flowing to a single host and port. We then use Mahalanobis distance during the detection phase to calculate the similarity of new data against the pre-computed profile. The detector compares this measure against a threshold and generates an alert when the distance of the new input exceeds this threshold.

The full technical details of the sensor have been reported in a publication:

> Ke Wang, Salvatore J. Stolfo. “Anomalous Payload-based Network Intrusion Detection”. Proceedings of Recent Advances in Intrusion Detection, *RAID*, Sept. 2004.

# 7. File System Anomaly Detection

FWRAP is a Host-based Intrusion Detection System that monitors file system calls to detect anomalous accesses. The system is intended to be used not as a standalone detector, but as one of a correlated set of host-based sensors. The detector has two parts, a sensor that audits file systems accesses, and an unsupervised machine learning system that computes normal models of those accesses. We developed an architecture for the file system sensor and implemented it on Linux using the FiST file wrapper technology and measured results of the anomaly detector applied to experimental data acquired from this sensor. FWRAP employs the Probabilistic Anomaly Detection (PAD) algorithm previously reported in our work on Windows Registry Anomaly

Detection and utilized in the MCP study. FWRAP represents a general approach to anomaly detection. The detector is first trained by operating the host computer for some amount of time and a model specific to the target machine is automatically computed by PAD, intended to be deployed to a real-time detector. In the following paper, we fully detail the feature sets used to model file system accesses, as well as the performance results of a set of experiments using the sensor while attacking a Linux host with a variety of malware exploits:

> Salvatore J. Stolfo, Shlomo Hershkop, Linh H. Bui, Ryan Ferster, and Ke Wang. "Unsupervised Anomaly Detection in Computer Security and An Application to File System Accesses". Proceedings of the International Symposium on Methodologies for Intelligent Systems, 2005.

Copies of both cited papers have been included in the final report transmitted.