Host-based Anomaly Detection via Resource Usage Signatures

Sahin Albayrak[1], Seyit Ahmet Çamtepe[1],
Matthew Edman[2], and Bülent Yener[2⋆]

[1] Technische Universität Berlin, Berlin, Germany
{Sahin.Albayrak,Ahmet.Camtepe}@dai-labor.de
[2] Rensselaer Polytechnic Institute, Troy, NY, USA
{edmanm2,yener}@cs.rpi.edu

Abstract. In this work we introduce a novel approach to host-based anomaly detection that utilizes techniques from tensor analysis to derive per-user behavioral signatures from higher-order datasets describing resource usage in a multi-user system. We show that a user's behavior can be characterized using easy-to-obtain system metrics, such as processor time consumption, memory allocation and disk I/O operations.
Our approach models the history of a user's resource usage metrics over all processes as a tensor and applies the Tucker3 multiway analysis model to capture the principal components of the tensor that can uniquely identify a user. To demonstrate the efficacy of our approach, we identify and collect usage statistics on six types of system resources from a remote-access university computer system over a period of 25 days. We use the collected real-world data to quantitatively evaluate stability of a user's resource fingerprint as well as the "distance" between the fingerprints of any two users. We further use the data to define parameters for a simulation model and evaluate our system's performance under various synthetic scenarios.

1 Introduction

Anomaly detection, in its broadest sense, is the study of identifying observations of a system that do not conform to the expected behavior. It has been applied in a diverse range of application domains, from credit card fraud [1] to anomaly detection in medical test data [2]. The subset of anomaly detection as pertaining to computer intrusion detection systems aims to distinguish normal, expected user or network behavior from abnormal behavior resulting from an attack on, or successful intrusion into, a computer system. Such systems proposed to date have been based on various techniques, such as sequence analyses of system calls [3] or a statistical analysis of network connections [4].

The approach presented in this paper is based on the intuition that we as computer users perform many of the same tasks each time we user our computers.

⋆ This work was supported by EU Marie Curie Fellowship Award: PIIF-GA-2008-221077, DMASD4CA

For example, we log in to the operating system, check our email, read the latest news on the Web, etc. We even do many of these tasks at roughly the same time each day. These patterns of daily application usage thus result in discernible fluctuations in the amount of system resources a user consumes (e.g., CPU time, memory consumption, etc).

Our work shows that it is possible to quantifiably characterize individual users based on patterns of resource usage and to develop unique signatures for each user. The signatures of "normal" behavior can then be incorporated into an anomaly detection system in order to identify periods when a user's behavior deviates significantly from their expected profile, potentially indicating an intrusion. We note, however, that the goal of our system is not to identify the form or source of a suspected intrusion. Rather, we seek only to raise an alert for further inspection by a system administrator or other entity when a user's behavioral profile is significantly altered.

Of course, there are times when we perform tasks that deviate from our normal behavior, such as performing computationally expensive experiments. Further, the resource usage information on a system with many users can be quite large. Thus, an anomaly detection system based on resource usage must be efficient enough to operate on large amounts of data and resilient enough to tolerate noise in the data.

The remainder of this paper is structured as follows. We will first provide some background on anomaly detection as it pertains to intrusion detection systems, review some related work in the field and discuss the key factors that set our approach apart from others. Next, in Section 2, we will describe the methodology employed to collect the real-world data necessary to validate the effectiveness in our approach. In Section 2.2, we describe the tensor analysis techniques used to project the higher-order resource usage data onto a lower dimension. Section 3 describes our approach for extracting signatures from the collected resource usage data. Next, in Section 5, we provide the results and analysis of our simulation. Finally, in Section 6, we summarize and discuss our results and findings.

1.1 Related Work

The general field of anomaly detection is indeed vast and a full treatment of the literature is outside the scope of this paper. We refer the interested reader to more comprehensive surveys on the subject for further discussion and references [5, 6]. In this section we will provide a brief background and overview of the work most closely related to our own.

Techniques for anomaly detection in computer systems can be broadly categorized into host-based and network-based approaches. Network-based approaches, as the name suggests, look for patterns in incoming and/or outgoing packets and attempt to identify attacks or other system abuse. Host-based anomaly detection systems, on the other hand, most often use system logs, audits and other process information in order to identify abuse occurring on the system or systems being monitored. In both approaches, techniques used to identify outliers within the

collected data have ranged from simple rule-based techniques [7] to statistical modeling [3] and neural networks [8].

The prior work closest to our own in concept is the Ayaka system described by Sugaya et al [9, 10]. Like our approach, Sugaya et al. characterize users based on patterns of resource usage. Their system uses k-means clustering to quantize observed CPU, memory and network usage values into a fixed number of clusters (eight, in their experiments). An ergodic hidden Markov model is then derived to model the expected per-user resource behavior based on a "known-good" observation period. Finally, an online process, again using hidden Markov models, is used to detect anomalies in the observed resource usage as compared to the models derived during the known-good period.

A key disadvantage of the Ayaka system is that it is sensitive to a number of parameters for which appropriate values may be difficult to forecast. In their evaluation, the authors chose a number of potential values for the system's parameters, evaluated each under a particular attack scenario (i.e., a successful SQL injection attack) and then chose the parameters that yielded the best performance. It is not clear, however, that the pre-determined parameters extend to other system and attack models.

A more practical disadvantage of Ayaka is that it requires kernel-level hooks in the operating system for monitoring purposes. Thus, it is not a portable solution nor simple to implement in practice. Further, the authors acknowledge that monitoring performed by the kernel-level hooks introduce non-trivial additional computational overhead into *all* running processes.

1.2 Our Contributions

Our primary contribution over the previous work discussed above is the introduction of a novel host-based technique to detect anomalous user behavior based on patterns of resource usage. We utilize techniques from tensor analysis to derive per-user behavioral signatures from higher-order datasets describing resource usage in a multi-user system.

A key advantage of our approach is that the data required for analysis and detection is easily obtained using commonly available system commands (discussed further in Section 2) and does not introduce significant performance overhead. While the resource usage data may be large, the resulting user signatures have very small storage requirements.

2 Sampling of Resource Usage Data

In this section, we describe our simple approach based on random sampling to collecting the data necessary to construct a three-way tensor for deriving resource-based signatures of user behavior.

2.1 Construction of Resource Usage Tensors

Consider a background monitoring process operating on the system to be observed, much like a software firewall or IDS application (e.g., Snort). In our approach, the monitoring process uses the following approach for collecting user resource data to be used for anomaly detection:

1. Let $t_s \geq 1$ be the duration of observation corresponding to a single tensor slice. The value of t_s can be chosen by a system administrator based on how frequently he or she wishes to verify observed user signatures. Also, let $t_e = 0$ be the number of seconds elapsed during a sampling interval.
2. The monitoring application chooses $t_d \in [1, t_s - t_e]$ uniformly at random and sleeps for t_d seconds.
3. At the expiration of t_d seconds, the application creates a snapshot of all running processes on the system. For each process in the snapshot, we extract multiple resource metrics resulting in an 8-tuple of the form

$$(uid, pid, cputime, pmem, inblk, oublk, inpkt, oupkt).$$

   A description of each field is given in the following table:

Feature	**Description**
pid	Unique identifier of a running process.
uid	The identifier of the user running the process.
cputime	CPU time consumed over the duration of the process.
pmem	Percent of total system memory consumed by the process.
inblk	Number of blocks read from disk.
oublk	Number of blocks written to disk.
inpkt	Number of network packets received.
oupkt	Number of network packets sent.

   Our approach is also flexible enough to support adding additional metrics to the monitoring process, if desired, simply by expanding the 8-tuple to an n-tuple.
4. We set $t_e = t_e + t_d$ and Steps 2 and 3 are repeated until $t_e = t_s$ seconds have elapsed. At the expiration of t_s seconds, we have an array of between 1 and t_s total snapshots, depending on the values randomly chosen for t_d in each iteration of the sampling.
5. For each process snapshot collected, the monitoring process aggregates the total resources consumed over all observed processes owned by each individual user and forms a matrix A wherein each row i corresponds to a single user and each column j is a resource type. An entry $A_{i,j}$ in this matrix is the total of resource j consumed by all processes for user i over the previous time period t_s. If a user i had no observed process activity, then $A_{i,j} = 0 \forall j$. We show in Section 5 that this does not adversely impact the accuracy of the user's fingerprint.

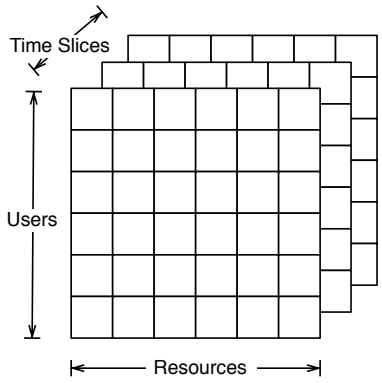


Fig. 1. General structure of a tensor of resource usage metrics. Each row represents a single observed user, each column is a resource type extracted from process tables and each slice corresponds to one complete sampling interval.

6. The resulting matrix corresponding to the current slice is appended to the resource tensor. A representation of the structure of this tensor is shown in Figure 1.
7. The monitoring process then returns to Step 1 and repeats the above process. Each iteration of this process appends one slice to the resource usage tensor.

In the next section, we will describe the techniques we apply to analyze the 3-way tensor constructed via the above process.

2.2 Data Analysis Techniques

We consider both 2-way and multiway data analysis to explore the data collected from a time sharing system. Multiway methods such as Tucker3 are generalizations of singular value decomposition (SVD) to high-order datasets. Both methods give us a substantially smaller set representing the original data. Compared to SVD, they generate component matrices containing singular vectors for each mode. On the other hand, Tucker3 represent the relationships between the modes with a core tensor as opposed to the presence of core matrix in SVD. We start with two-mode analysis, and then introduce three-mode techniques briefly.

A high-order tensor is a multiway dataset represented as $A \in R^{n_1 \times n_2 \times \cdots \times N}$, where $N > 2$ [11, 12]

Singular Value Decomposition We use SVD to model per user behavior represented by a matrix (say A) over time with respect to resource usage. Each entry A_{ij}, of data matrix $A \in R^{nxm}$, shows the resource i used by the user at time$j$. The formal definition of SVD is $A = USV*$, where $U \in R^{nxm}$ and $V \in R^{mxm}$ are orthogonal matrices containing left and right singular vectors, respectively and $S \in R^{mxm}$ is a diagonal matrix with singular values on the diagonal as

shown in Figure 2(a). Singular values are the weights of the component vectors in matrix U. We choose the greatest singular values and the corresponding left singular vectors through rank reduction to represent the data in a smaller space.

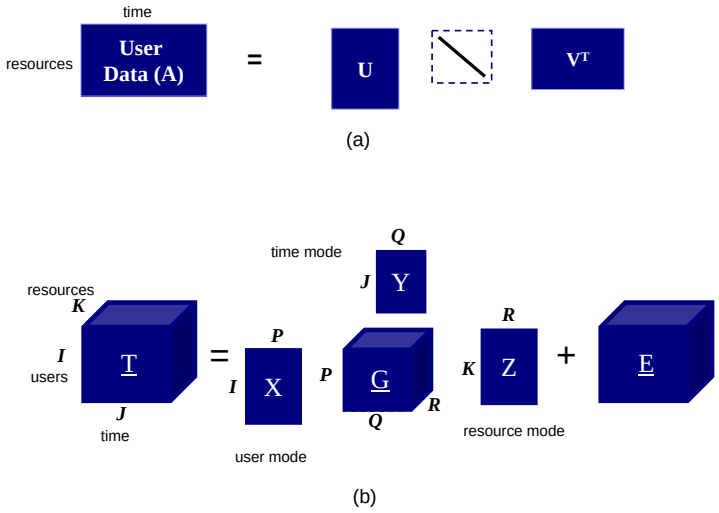


Fig. 2. Data modeling and analysis techniques, where (a) shows modeling of data using bilinear model SVD, and (b) shows 3-way modeling by the Tucker3 model. We used SVD on user component matrices of T to construct spectral signatures. Tucker 3 is used to discover outliers in user mode.

Tucker3 Tucker3, also called three-mode principal component analysis, is proposed by Ledyard R. Tucker and is one of the models in Tucker family (Tucker1, Tucker2, Tucker3) [13]. When we employ Tucker3 model on a tensor $T \in R^{n \times m \times l}$, we obtain three component matrices as shown in Figure 2.b. Relationships between different modes are represented by a core tensor G where each entry, G_{ijk}, can be nonzero. Tucker decomposition of T is expressed as follows:

$$T_{ijk} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} G_{r_1 r_2 r_3} X_{i r_1} Y_{j r_2} Z_{k r_3} + Eij\ k$$

where R_1, R_2 and R_3 indicate the number of components in mode 1,mode 2, mode 3, respectively. $X_{R_n \times R_1}$, $Y_{R_m \times R_2}$ and $Z_{R_l \times R_1}$ are the component matrices, $G \in R^{r_1 \times r_2 \times r_3}$ is the core tensor and E_{ijk} is the error term.

3 Constructing & Verifying User Signatures

We now present our approach for deriving a compact, quantitative signature of a user's resource usage behavior from data obtained using the periodic sampling approach described in Section 2. The approach applies multiway analysis techniques described in Section 2.2 to extract a *baseline signature vector* describing a user's baseline signature from which anomalous deviations in future behavior can be detected.

3.1 Deriving Baseline Signatures

We make the assumption that a new user's account on a system is initially not compromised for a period of time. It is during this time that we establish the initial baseline behavioral profile of a user. We recognize that it is sometimes possible for attacker to create new system accounts solely for his or her own malicious purposes; however, the ability to create new user accounts is typically restricted to `root` or other "superusers" and if the attacker has `root`-level privileges, then there is little defense for the system at that point. Thus, we believe our assumption that there exists a period of observation during which the system can learn a true baseline signature for a user without the presence of an attacker is reasonable.

Recall that the data collection in Section 2 yields a 3-way tensor $\mathbf{T} \in \mathbb{R}^{r \times n \times t}$, where r is the number of resource types extracted from process data, n is the number of observed users and t is total number of tensor slices. From the baseline tensor $\mathbf{T}$, we can apply SVD to derive the vector of spectral coefficients used to model a user's resource usage. Let

$$f(\mathbf{T}_{i\ldots j}, u) = \mathbf{v}^u = \begin{pmatrix} c_1^u \\ c_2^u \\ \vdots \\ c_r^u \end{pmatrix} \tag{1}$$

be the SVD function applied to slices $i, i+1, \ldots, j$ of the baseline tensor $\mathbf{T}$ for user u, where vector $\mathbf{v}^u \in \mathbb{R}^m$ is the derived vector of spectral coefficients.

We first derive a series of columnar *signature estimate vectors*, which are intended to be a first approximation of the user's resource usage fingerprint. Let $b \in [1, t]$ be the total number of signature estimate vectors to use for baseline signature computation, where b is configurable by the system administrator. For each user u in tensor $\mathbf{T}$, we compute the following series of vectors:

$$\{f(\mathbf{T}_{1\ldots t-b}, u), f(\mathbf{T}_{1\ldots t-b+1}, u), \ldots, f(\mathbf{T}_{1\ldots t}, u)\}$$

$$= \{\mathbf{v}_1^u, \mathbf{v}_2^u, \ldots, \mathbf{v}_b^u\}$$

$$= \left\{ \begin{pmatrix} c_{1,1}^u \\ c_{2,1}^u \\ \vdots \\ c_{r,1}^u \end{pmatrix}, \begin{pmatrix} c_{1,2}^u \\ c_{2,2}^u \\ \vdots \\ c_{r,2}^u \end{pmatrix}, \ldots, \begin{pmatrix} c_{1,b}^u \\ c_{2,b}^u \\ \vdots \\ c_{r,b}^u \end{pmatrix} \right\} \tag{2}$$

From the series of signature estimate vectors $\mathbf{v}_1^u, \mathbf{v}_2^u, \ldots, \mathbf{v}_b^u$ for user u, we define the u's final baseline signature vector $\mathbf{s}_u$ as

$$\mathbf{s}_u = \begin{pmatrix} s_1^u \\ s_2^u \\ \vdots \\ s_r^u \end{pmatrix} = \begin{pmatrix} \frac{1}{b}\sum_{i=1}^{b} c_{1,i}^u \\ \frac{1}{b}\sum_{i=1}^{b} c_{2,i}^u \\ \vdots \\ \frac{1}{b}\sum_{i=1}^{b} c_{r,i}^u \end{pmatrix} \tag{3}$$

In words, the final baseline signature vector $\mathbf{s}_u$ for user u is composed of row-wise averages from the series of estimate vectors $\mathbf{v}_1^u, \mathbf{v}_2^u, \ldots, \mathbf{v}_b^u$ for u.

In addition to the baseline signature vector, we also define an allowable signature deviation parameter δ_u for each user u that specifies how much a user's behavior can differ from its baseline signature before being flagged as anomalous. Let $\alpha \in [0, 1]$ be an administrator-controlled "sensitivity" parameter. For a user u, the allowable deviation δ_u is defined to be

$$\delta_u = \alpha \cdot \sqrt{\frac{1}{b}\sum_{i=1}^{b}\sum_{k=1}^{r}(s_k^u - c_{k,i}^u)^2}. \tag{4}$$

Finally, the output of this initial signature establishment is a 3-tuple $(u, \mathbf{s}_u, \delta_u)$ for each user u containing the user's expected signature vector and the maximum tolerated deviation from the expected signature. The 3-tuple for each user is then stored on the system and subsequently used for detecting anomalous signatures, described in the next section.

3.2 Detecting Anomalous Signatures

After constructing baseline signatures for each user in the system, we have the tuple $(u, \mathbf{s}_u, \delta_u)$ for each user u in the system. Let $\tilde{\mathbf{T}}$ be a resource tensor of observed user behavior, collected in the same manner as the baseline tensor used for establishing user resource signatures in the previous section. After a tensor slice is appended to $\tilde{\mathbf{T}}$, the monitoring process computes the observed resource signature $\tilde{\mathbf{s}}_u$, where $\tilde{\mathbf{s}}_u = f(\tilde{\mathbf{T}}, u)$, for each user u in the system.

Then, the monitor computes the L2 distance between the pre-established baseline signature vector for u and the observed signature vector. If

$$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u) \geq \delta_u,$$

the monitor concludes that the observed behavior is anomalous and should be forwarded to an administrator for further review.

4 Experimental Results

In this section, we describe real-world data collected from a university computing system in order to evaluate the effectiveness of our approach. In the next section we also describe a simulation study we built based on the real-world data, which enabled us to further evaluate our system under additional, particular scenarios that we were not able to observe on a "live" system.

4.1 Data Collection and Analysis

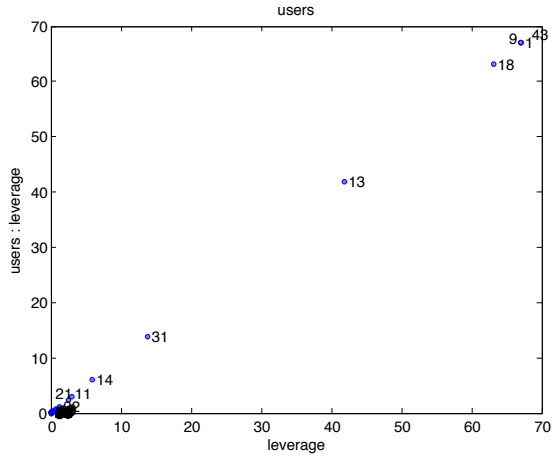
We implemented a simple background process to collect actual per-user resource data and construct a 3-way tensor as described in the previous sections. The collector was written in Python and ran on a remote-access university computer system at Rensselaer Polytechnic Insitute running the FreeBSD 6.3 operating system. The system is typically used by faculty, staff and students for everything from everyday tasks like email and SSH, to more specialized tasks, such as running experiments.

We ran the data collector for a continuous period of 26 days in August 2009[3]. The sampling configuration parameters chosen were a tensor slice interval of $t_s = 14400$ seconds (4 hours) and a maximum process table sampling interval of $t_d = 3600$ seconds (1 hour), resulting in a total of 156 tensor slices.

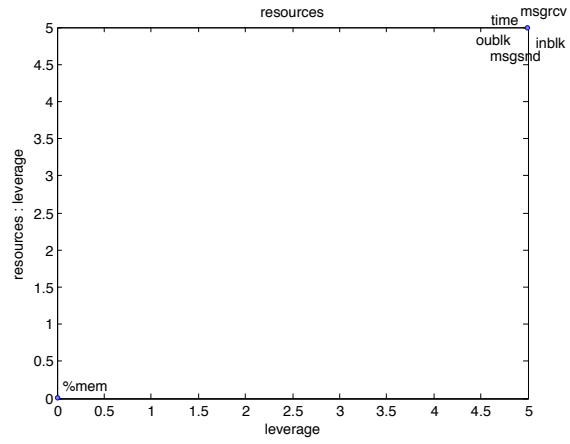
During the observation period, we collected data on a total of 68 unique users. Some of the users were always-on system users (e.g., `root` and `daemon`), while most were typical interactive users. The first analysis we performed on the data was to evaluate whether the user resource distribution for each user and resource type was consistent across the period of evaluation. We extracted a 50-sample sliding window from the original data as the baseline distribution sample and the remainder as the test sample. As we "slid" the baseline window across the sample set, we repeatedly performed a two-sample Kolmogorov-Smirnov test to evaluate the null hypothesis that the base and test samples were from the same distribution. We used a significance level of $\alpha = 0.05$ in all tests.

The results in Table 1 indicate the percentage of trials for each resource type wherein we accept the null hypothesis and conclude that the base and test samples are from the same distribution. We observed that the users with the least consumption had the highest success rate in the two-sample KS tests, likely because their resource usage levels were 0 in almost all sample slices. In nearly all cases, it appears that deriving a distribution's parameters from a sample set and then testing subsequent samples against the theoretical distribution seems to yield very poor results. We see the failure of this naïve approach to profiling user

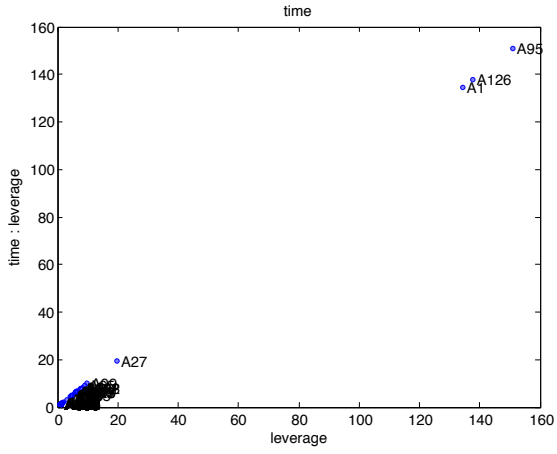
[3] Our initial plan had been to collect data for 30 days, but the experiment ended four days prematurely due to a hardware failure on the system. We do not believe this affects the accuracy of our results.



(a)



(b)



(c)

Fig. 3. We fit a Tucker 3 model with core G dimensions 5x5x5 with an accuracy of 98.63%. The influence analysis in *user*, *resource*, and *time* modes are shown in (a), (b), and (c), respectively.

u	pmem	cputime	inblk	outblk	inpkt	outpkt
1	22.64	56.60	100.0	28.30	51.89	100.0
7	100.0	100.0	100.0	100.0	100.0	100.0
9	12.26	23.59	100.0	100.0	23.59	24.53
11	100.0	100.0	100.0	100.0	100.0	100.0
13	4.72	13.21	100.0	99.06	24.53	24.53
14	0.00	0.00	100.0	0.00	0.00	0.00
44	68.87	100.0	100.0	100.0	100.0	100.0

Table 1. Results from a two-sample Kolmogorov-Smirnov test. Each entry indicates the percentage of trials for each resource type wherein we accept the null hypothesis and conclude that the base and test samples are from the same distribution.

resource behavior as motivation for the more sophisticated approach described in this paper.

In Figure 3 we show the Tucker 3 model fitting and analysis results. The user mode analysis in Figure 3(a) indicates that some users including 1, 9 are outliers. However, since Tucker3 is an unsupervised technique, it does not provide us information on whether or not there is an "anomaly." Thus, it provides the motivation and justification of our work presented in next sections. Analysis of the resource mode shows that five out of six resources have a high influence in the modeling of the data (upper right corner). Finally, the time mode shows that the information contained in time slices are quite stable except a few ones.

In summary, from the tensor analysis we learn that (1) for anomaly detection we need to build new techniques since unsupervised learning methods cannot discriminate between outliers and anomalies, (2) the choice of resources we used in this work is well justified, (3) the duration for which we collected the data the underlying timesharing system is quite stable.

4.2 Establishing Baseline Signatures

Assuming that a new user's account on a system is initially not compromised for a period of time, we are able to establish the initial baseline behavioral profile of a user. Intuitively, a longer signature establishment period will result in a more accurate baseline signature. In the absence of labeled data where anomalies and normal user behavior are well-separated by an expert, our approach aims to build the baseline signatures computationally and train our algorithm based on the simple assumption above.

We applied the algorithm described in Section 3 to derive the baseline signatures for all stable users based on the first 151 slices of tensor data which includes the outlier slices 1, 95 and 126. Out of the 68 users, seven had a "stable" fingerprint including some outlying users detected by the tensor analysis that we picked for further individual analysis.

An important aspect of this system to consider is not just each user's baseline signature vector, but also how much each user's signature differs from every other user. Such a metric allows us to examine how "unique" a user's behavioral profile

is when compared to other users observed at the same time. For that purpose, we create a *signature distance matrix* $D_{n,n}$ as

$$D_{n,n} = \begin{bmatrix} d(\mathbf{s}_1, \mathbf{s}_1) & d(\mathbf{s}_1, \mathbf{s}_2) & \cdots & d(\mathbf{s}_1, \mathbf{s}_n) \\ d(\mathbf{s}_2, \mathbf{s}_1) & d(\mathbf{s}_2, \mathbf{s}_2) & \cdots & d(\mathbf{s}_2, \mathbf{s}_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(\mathbf{s}_n, \mathbf{s}_1) & d(\mathbf{s}_n, \mathbf{s}_2) & \cdots & d(\mathbf{s}_n, \mathbf{s}_n) \end{bmatrix}, \tag{5}$$

where,

$$d(\mathbf{s}_u, \mathbf{s}_v) = \sqrt{\sum_{i=1}^{r} (s_i^u - s_i^v)^2}. \tag{6}$$

for users u and v. In other words, each entry in $D_{n,n}$ is the pairwise L2 distance between two user signature vectors. The resulting matrix describes the similarity (or dissimilarity) of signatures between each pair of users observed on the system.

We computed a $D_{7,7}$ signature distance matrix for the outlying users using their derived signatures. The results and analysis for each stable user are given in Table 2. The table shows that most of the user signatures exhibited excellent separation from the rest. The next section discusses how accurate these derived signatures are when used to identify their respective users based on resource usage statistics.

u	1	7	9	11	13	14	44	*min*	*max*	*mean*	*stddev*
1	0	1.2431	1.0375	0.7511	0.4385	0.7369	0.6916	0.4384	1.2431	0.8164	0.2828
7	1.2431	0	0.2523	0.5573	0.8818	0.5887	0.6319	0.2523	1.2431	0.6925	0.3363
9	1.0375	0.2523	0	0.3133	0.6485	0.3478	0.3875	0.2523	1.0375	0.4978	0.2976
11	0.7511	0.5573	0.3133	0	0.3392	0.0828	0.0753	0.0753	0.7511	0.3532	0.2654
13	0.4384	0.8818	0.6485	0.3392	0	0.3246	0.2705	0.2705	0.8818	0.4838	0.2364
14	0.7369	0.5887	0.3478	0.0828	0.3246	0	0.1012	0.0828	0.7369	0.3637	0.2606
44	0.6916	0.6319	0.3875	0.0753	0.2705	0.1012	0	0.0753	0.6916	0.3597	0.2610

Table 2. Baseline signature distance matrix $D_{n,n}$ for $n = 7$ of the observed users. In general, we saw an excellent separation between most of the stable user fingerprints, supporting our claim that user behavior can be profiled using only basic resource usage information.

4.3 Testing for Anomalies

Since we used the first 151 slices of tensor data for establishing baseline signatures, we used the remaining five slices (non outliers) for testing the accuracy of the derived baseline signatures. From the five test slices, we formed a tensor

$\tilde{\mathbf{T}}$ and extracted test signatures $\tilde{\mathbf{s}}_u$ for each user u according to the technique described in Section 3.

After extracting the signature of each observed user, we then computed the distance $d(\mathbf{s}_u, \tilde{\mathbf{s}}_u)$ between each user's baseline signature and their observed test signature over five tensor slices. The computed distances are given in Table 3.

u	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^1)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^2)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^3)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^4)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^5)$	δ_u
1	0.0048	0.0145	0.0144	0.0144	0.0144	0.0044
7	0.0002	0.0002	0.0002	0.0002	0.0003	0.0002
9	0.0003	0.0003	0.0003	0.0003	0.0003	0.0001
11	0.1195	0.1195	0.1195	0.1195	0.1195	0.1919
13	0.0000	0.0000	0.0001	0.0001	0.0010	0.0001
14	0.0044	0.0057	0.0067	0.0077	0.0086	0.0029
44	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Table 3. Distance of observed resource usage signatures from the users' expected baseline signatures as compared to the maximum allowed deviation from the baseline. Of the five users above, three had observed signatures that exceeded their maximum allowed deviation (δ_u).

Given the computationally established baseline signatures of the remaining users, the table above shows that anomalous users 1 and 14 clearly deviated from the expected behavior and would be flagged as anomalous. Users 7, 11, 13 and 44 were mostly within their allowed signature deviation range during the observed tensor slices. The distance between user 9's baseline and observed signatures was quite small, but would still have been flagged. Increasing the value of α used to compute δ_u, in this case, would have reduced false positives without adversely impacting accuracy.

5 Building a Simulation Model from Measurements

In this section, we describe the development of a simulator that allowed us to evaluate more specific user scenarios than we were able to observe in our real-world data collection, while still maintain realistic user behavior. Specifically, we wanted to evaluate whether the periodicity of a user's behavior had an impact on the accuracy of our approach. That is, can we develop signatures of users who are always active, sometimes active at regular intervals or only active aperiodically? We also wanted to evaluate unstable users–users whose resource usage while active is randomly distributed and follows no discernible pattern.

5.1 Simulation Model

We developed a simulator that generated resource usage tensors for a configurable number of users. Each simulated user had an activity schedule during

which they had a predefined probability $p \in [0, 1]$ of being active for each hour of a simulated day. This allowed us to model various activity behaviors, such as always-on users, users who are only active during the day and users who are active randomly throughout the day.

For the various resource types, we typically modeled the CPU time and memory consumption using exponential distributions, while disk and network I/O were most often modeled using Poisson distributions. For some users and resource types, a Gamma distribution or constant value were the optimal fit for their observed profile. The specific parameters for each distribution for a particular user were obtained by fitting a distribution to the observed data for each user-resource pair. Thus, the simulated resource usage for each user in the simulation was representative of real-world resource usage profiles. The specific user activity schedules, activity probabilities and distribution parameters for each simulated user is given in Appendix A.

We generated a three-way tensor with nine users and 180 time slices, representing a 30 day observation period. Users 1, 2 and 3 were always-on users with varying resource distribution parameters. Users 4, 5 and 6 were typical interactive users who were only active during a portion of the day (e.g., during normal work hours). The final three users were modeled as "unstable" users who had no discernible resource behavior patterns; their resource usage was generated uniformly at random. Users 7 and 8, in particular, were given similar uniform distribution parameters, but disparate activity times.

The baseline signature analysis results for the tensor of simulated users is given in Table 4. In general, the results show that our approach to signature derivation is largely unaffected by underlying patterns of when the user is online, or lack thereof. Instead, it depends only on a user's resource behavior during the periods it is online. This reinforces our earlier statement that inserting a row containing all zeros in slices for which a particular user had no observed process activity does not significantly affect the derivation of its fingerprint.

u	1	2	3	4	5	6	7	8	9	min	max	avg	$stddev$
1	0.0000	1.1029	0.7774	0.5985	0.6038	0.7103	0.7404	0.7404	0.7329	0.5985	1.1029	0.7508	0.1566
2	1.1029	0.0000	0.6836	0.8842	0.8738	0.4638	0.7177	0.7177	0.7570	0.4638	1.1029	0.7751	0.1857
3	0.7774	0.6836	0.0000	0.2756	0.2557	0.3945	0.0877	0.0877	0.2293	0.0877	0.7774	0.3489	0.2570
4	0.5985	0.8842	0.2756	0.0000	0.0292	0.5039	0.2317	0.2317	0.2900	0.0292	0.8842	0.3806	0.2682
5	0.6038	0.8738	0.2557	0.0292	0.0000	0.4960	0.2139	0.2139	0.2767	0.3945	0.7103	0.4776	0.1032
6	0.7103	0.4638	0.3945	0.5039	0.4960	0.0000	0.4017	0.4017	0.4486	0.0292	0.8738	0.3704	0.2702
7	0.7404	0.7177	0.0877	0.2317	0.2139	0.4017	0.0000	0.0000	0.1508	0.0000	0.7404	0.3180	0.2792
8	0.7404	0.7177	0.0877	0.2317	0.2139	0.4017	0.0000	0.0000	0.1508	0.0000	0.7404	0.3180	0.2792
9	0.7329	0.7570	0.2293	0.2900	0.2767	0.4486	0.1508	0.1508	0.0000	0.1508	0.7570	0.3795	0.2444

Table 4. Statistics on the distance between baseline signature vectors for all simulated users. The results show that our approach to signature derivation is largely unaffected by underlying patterns of when the user is online, or lack thereof. Instead, it depends only on a user's resource behavior during the periods it is online.

5.2 Testing for Anomalies

We then used the remaining five tensor slices as test vectors to evaluate the accuracy of the derived fingerprints. The results of the evaluation are shown in Table 5. Our approach successfully identified each of the always-on users with high accuracy. Additionally, we were able to identify two out of the three occasional users (user 4, 5 and 6).

u	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^1)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^2)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^3)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^4)$	$d(\mathbf{s}_u, \tilde{\mathbf{s}}_u^5)$	δ_u
1	0.0003	0.0005	0.0005	0.0006	0.0005	0.0005
2	0.0480	0.0236	0.0494	0.0244	0.0107	0.0347
3	0.0025	0.0030	0.0041	0.0064	0.0022	0.0040
4	0.0028	0.0028	0.0013	0.0013	0.0013	0.0020
5	0.0067	0.0068	0.0027	0.0027	0.0027	0.2939
6	0.3320	0.2836	0.2836	0.2836	0.2836	0.0048
7	0.0978	0.0978	0.0788	0.0788	0.0788	0.0869
8	0.3643	0.3643	0.3643	0.3643	0.3643	0.3643
9	0.4230	0.4230	0.4230	0.4230	0.4230	0.4230

Table 5. Distance of test signature vectors from the expected baseline signature vectors, as well as the maximum allowed deviation δ_u computed from the baseline tensor with $\alpha = 1.0$.

As expected, the users with resource usage generated uniformly at random were unable to be correctly identified. For such unstable users, no consistent baseline signature can be derived and thus cannot be identified with our system.

6 Conclusions

Through both real-world experimentation and simulation, we demonstrated a novel approach to host-based anomaly detection that utilizes techniques from tensor analysis to derive per-user behavioral signatures from higher-order datasets describing resource usage in a multi-user system. We showed that a user's behavior can be characterized using easy-to-obtain system metrics, such as processor time consumption, memory allocation and disk I/O operations.

In the examples we described in this paper, we used a total of six different types of system resources to derive user fingerprints. It is important to note, however, that our approach can easily be extended to include any additional numerical resource metrics (e.g., number of running processes, context switches, interval between commands, etc). The only changes required to the system described is one additional column in each tensor slice for each additional resource type monitored.

The primary potential limitation of our spectral-based signature approach is the time required to learn an accurate approximation of a user's signature.

In our real-world data analysis, we found that it was difficult to distinguish between many of the real-world users who were only briefly online during our data collection period. Establishing optimal bounds on the minimum time required to establish a user's baseline behavioral fingerprint is an interesting potential topic for future research.

References

1. Ghosh, S., Reilly, D.: Credit card fraud detection with a neural-network. In: Proceedings of the 27th Annual Hawaii International Conference on System Science. Volume 3. (1994)
2. Suzuki, E., Watanabe, T., Yokoi, H., Takabayashi, K.: Detecting interesting exceptions from medical test data with visual summarization. In: Proceedings of the 3rd IEEE International Conference on Data Mining. (2003) 315–322
3. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A sense of self for unix processes. In: Proceedings of the 1996 IEEE Symposium on Security and Privacy. (1996) 120–128
4. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of network traffic for detecting novel attacks. In: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2002) 376–385
5. Patcha, A., Park, J.M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks **51**(12) (2007) 3448–3470
6. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Computing Surveys **41**(3) (2009) 1–58
7. Branch, J., Szymanski, B., Giannella, C., Wolff, R., Kargupta, H.: In-network outlier detection in wireless sensor networks. In: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems. (2006) 51–58
8. Ghosh, A.K., Wanken, J., Carron, F.: Detecting anomalous and unknown intrusions against programs. (1998) 259–267
9. Sugaya, M., Ohno, Y., Nakajima, T.: Lightweight anomaly detection system with HMM resource modeling. International Journal of Security and Its Applications **3**(3) (2009) 35–54
10. Ohno, Y., Sugaya, M., van der Zee, A., Nakajima, T.: Anomaly detection system using resource pattern learning. In: Software Technologies for Future Dependable Distributed Systems. (2009) 38–42
11. Kolda, T.G.: Orthogonal tensor decompositions. SIAM Journal on Matrix Analysis and Applications **23**(1) (2001) 243–255
12. Kolda, T.G., Bader, B.W.: MATLAB tensor classes for fast algorithm prototyping. Technical Report SAND2004-5187, Sandia National Laboratories, Livermore, CA (October 2004)
13. Tucker, L.R.: Some mathematical notes on three-mode factor analysis. Psychometrika **31**(3) (September 1966) 279–311

A Simulated User Configurations

This Appendix contains the activity schedules and distribution parameters used to generate the resource usage distributions for each user in our simulation. Users

are grouped into three types. Type I users have stable resource usage profiles and are always online (e.g., `root`, `daemon` or other system users). Type II users are normal interactive users. Type III users have no particular behavioral patterns and are their resource usage is generated uniformly at random within pre-defined intervals. For Type I and Type II users, we based their distributions types and parameters on the users we observed in the data collected in Section 4.

Type I Users

User 1 (Stable, always-on)	
0000 to 2359 ($P_{active} = 1.0$)	
`cputime`	$Gamma(k = 29730.74, \theta = 1.04)$
`pmem`	$Exponential(\lambda = 0.19)$
`inblk`	$Poisson(\lambda = 223.41)$
`outblk`	$Poisson(\lambda = 6399.06)$
`inpkt`	$Poisson(\lambda = 14386.68)$
`outpkt`	$Poisson(\lambda = 65770.78)$

User 2 (Stable, always-on)	
0000 to 2359 ($P_{active} = 1.0$)	
`cputime`	$Exponential(\lambda = 48.60)$
`pmem`	$Const(k = 0.2)$
`inblk`	$Poisson(\lambda = 0.006)$
`outblk`	$Const(k = 0.0)$
`inpkt`	$Poisson(\lambda = 129.95)$
`outpkt`	$Poisson(\lambda = 9.03)$

User 3 (Stable, always-on)	
0000 to 2359 ($P_{active} = 1.0$)	
`cputime`	$Exponential(\lambda = 0.97)$
`pmem`	$Exponential(\lambda = 0.24)$
`inblk`	$Poisson(\lambda = 0.28)$
`outblk`	$Poisson(\lambda = 79.49)$
`inpkt`	$Poisson(\lambda = 739.40)$
`outpkt`	$Poisson(\lambda = 567.54)$

Type II Users

User 4 (Stable, periodic)	
0800 to 1800 ($P_{active} = 0.65$)	
`cputime`	$Exponential(\lambda = 0.46)$
`pmem`	$Exponential(\lambda = 0.24)$
`inblk`	$Poisson(\lambda = 0.24)$
`outblk`	$Poisson(\lambda = 1.07)$
`inpkt`	$Poisson(\lambda = 2969.77)$
`outpkt`	$Poisson(\lambda = 3881.67)$

User 5 (Stable, periodic)	
0600 to 1759 ($P_{active} = 0.85$)	
`cputime`	$Exponential(\lambda = 0.43)$
`pmem`	$Exponential(\lambda = 0.71)$
`inblk`	$Poisson(\lambda = 0.08)$
`outblk`	$Poisson(\lambda = 73.05)$
`inpkt`	$Poisson(\lambda = 1729.30)$
`outpkt`	$Poisson(\lambda = 2201.37)$
1800 to 2200 ($P_{active} = 0.025$)	
`cputime`	$Exponential(\lambda = 0.43)$
`pmem`	$Exponential(\lambda = 0.71)$
`inblk`	$Poisson(\lambda = 0.08)$
`outblk`	$Poisson(\lambda = 73.05)$
`inpkt`	$Poisson(\lambda = 1729.30)$
`outpkt`	$Poisson(\lambda = 2201.37)$

User 6 (Stable, periodic)	
0000 to 0959 ($P_{active} = 0.015$)	
`cputime`	$Exponential(\lambda = 5.82)$
`pmem`	$Exponential(\lambda = 34.90)$
`inblk`	$Poisson(\lambda = 0.01)$
`outblk`	$Const(k = 0)$
`inpkt`	$Poisson(\lambda = 85.64)$
`outpkt`	$Poisson(\lambda = 55.91)$
1000 to 1859 ($P_{active} = 0.95$)	
`cputime`	$Exponential(\lambda = 5.82)$
`pmem`	$Exponential(\lambda = 34.90)$
`inblk`	$Poisson(\lambda = 0.01)$
`outblk`	$Const(k = 0.0)$
`inpkt`	$Poisson(\lambda = 85.64)$
`outpkt`	$Poisson(\lambda = 55.91)$
1900 to 2359 ($P_{active} = 0.035$)	
`cputime`	$Exponential(\lambda = 5.82)$
`pmem`	$Exponential(\lambda = 34.90)$
`inblk`	$Poisson(\lambda = 0.01)$
`outblk`	$Const(k = 0.0)$
`inpkt`	$Poisson(\lambda = 85.64)$
`outpkt`	$Poisson(\lambda = 55.91)$

Type III Users

User 7 (Unstable, periodic)	
0800 to 2359 ($P_{active} = 0.5$)	
`cputime`	$Uniform(min = 0, max = 2)$
`pmem`	$Uniform(min = 0, max = 10)$
`inblk`	$Uniform(min = 0, max = 100)$
`outblk`	$Uniform(min = 0, max = 100)$
`inpkt`	$Uniform(min = 0, max = 1000)$
`outpkt`	$Uniform(min = 0, max = 1000)$
User 8 (Unstable, periodic)	
0800 to 1959 ($P_{active} = 0.75$)	
`cputime`	$Uniform(min = 0, max = 2)$
`pmem`	$Uniform(min = 0, max = 10)$
`inblk`	$Uniform(min = 0, max = 100)$
`outblk`	$Uniform(min = 0, max = 100)$
`inpkt`	$Uniform(min = 0, max = 1000)$
`outpkt`	$Uniform(min = 0, max = 1000)$
User 9 (Unstable, periodic)	
0000 to 0759 ($P_{active} = 0.05$)	
`cputime`	$Uniform(min = 0.2, max = 1.5)$
`pmem`	$Uniform(min = 0, max = 2)$
`inblk`	$Uniform(min = 0, max = 25)$
`outblk`	$Const(k = 0.0)$
`inpkt`	$Uniform(min = 10, max = 100)$
`outpkt`	$Uniform(min = 1, max = 10)$
0800 to 1559 ($P_{active} = 0.5$)	
`cputime`	$Uniform(min = 0.2, max = 1.5)$
`pmem`	$Uniform(min = 0, max = 5)$
`inblk`	$Uniform(min = 0, max = 50)$
`outblk`	$Uniform(min = 0, max = 15)$
`inpkt`	$Uniform(min = 0, max = 100)$
`outpkt`	$Uniform(min = 0, max = 100)$
1600 to 2359 ($P_{active} = 0.05$)	
`cputime`	$Uniform(min = 0.0, max = 1.0)$
`pmem`	$Uniform(min = 0, max = 1.0)$
`inblk`	$Uniform(min = 0, max = 25)$
`outblk`	$Const(k = 0.0)$
`inpkt`	$Uniform(min = 80, max = 100)$
`outpkt`	$Uniform(min = 80, max = 100)$