Evaluating Software Sensors for Actively Profiling Windows 2000 Computer Users

Jude Shavlik
Computer Sciences Department
University of Wisconsin – Madison

Mark Shavlik
Michael Fahland
Shavlik Technologies
White Bear Lake, MN
{jude, mark, mikef}@shavlik.com

Abstract

We report on a new, on-going intrusion-detection project that empirically investigates the usefulness of "stealing" a small amount of CPU cycles (1%), main memory (16MB), and disk memory (100 MB) in order to continually gather and analyze dozens of fine-grained system measurements, such as network traffic, identity of the current programs executing, and the user's typing speed. The underlying scientific hypothesis is that a properly chosen set of measurements can provide a "fingerprint" that is unique to each user. Hence, such measurements could serve to help distinguish appropriate use of a given computer from misuse, especially by insiders.

Introduction

In an increasingly computerized and networked world, it is crucial to develop defenses against malicious insider activity in information systems. One promising approach is to develop computer algorithms that detect insiders who are inappropriately intruding on the computers of others. However, *intrusion detection* is a difficult problem to solve [DARPA99]. System performance cannot be adversely effected, false positives must be minimized, and intrusions must be caught (i.e., false negatives must be very low). The current state of the art in intrusion-detection systems is not good; false positives are much too high and successful detection is unfortunately too rare.

Intrusion-detection systems (IDS) can either (a) look for known attack patterns or (b) be "adaptive software" that is smart enough to monitor and learn how the system is supposed to work under normal operation versus how it works when misuse is occurring [LUNT93]. We are addressing approach (b). Specifically, we are empirically determining which sets of fine-grained system measurements are the most effective at distinguishing usage by the assigned user of a given computer from misusage by other "insiders" [DARPA99; NEUMANN99] within an organization.

Building on our expertise in Windows 2000 computer security and machine learning, we have written and are currently extending a prototype anomaly-detection system that creates statistical profiles of the normal usage for a given computer running Windows 2000. Significant deviations from normal behavior indicate that an intrusion is likely occurring. For example, if the probability that a specific computer receives 10 Mbytes/sec during evenings is measured to be

very low, then when our monitoring program detects such a high transfer rate during evening hours, it will suggest that an intrusion may be occurring.

This ability to create statistical models of individual computer's normal usage means that each computer's unique characteristics serve a protective role. Similar to how each person's antibodies can distinguish one's own cells from invading organisms, these statistical-profile programs can, as they gather data during the normal operation of a computer, learn to distinguish "self" behavior from "foreign" behavior. For instance, some people heavily use the mouse when interacting with their computer's interface, while others strongly prefer to use special keyboard characters like the control key. Should someone leave their computer unattended and someone else try to inappropriately access their files, the individual differences between people's computer usage will mean that our statistical-modeling program will quickly recognize this illegal access. (Our approach can also be used to detect abnormal behavior in computers operating as HTTP, FTP, and email servers. However, this project currently focuses on computers used by humans.)

In addition to determining informative, real-time measurements indicative of insider misuse, a secondary objective is to determine an effective method for combining the multiple, individual measurements into a single, composite score whose value is indicative of the current threat of insider misuse. A third objective is to determine how best to "condition," in the formal probabilistic sense, individual measurements on other measured properties. The final objective is to empirical compare Bayesian networks, neural networks, and decision trees on the task of using the measured properties to learn accurate formal models for identifying which person is currently using a given computer.

The ability to detect insider misuse is being evaluated by collecting data from multiple users, creating user profiles by analyzing "training" subsets of this data, and then experimentally judging the accuracy of applying these profiles to identify the user that produced each sample set of "testing" measurements. That is, we empirically judge the ability of induced user profiles to recognize which specific user produced a given set of measurements. We are measuring the tradeoff between the recognition and false-positives rates under various experimental conditions. The key scientific hypothesis being investigated is whether or not creating statistical models of user behavior can be used to accurately detect insider abuse. We are focusing our algorithmic development on methods that produce very low false-alarm rates, since a major reason system administrators ignore IDS systems is that they produce too many false alarms.

Scientific questions being addressed in this project include the following:

- Which system and user measurements are the most informative for recognizing appropriate use of a computer running Windows 2000?
- What is the best method for combining multiple, individual measurements into a *single* score indicative of the current insider threat?
- What are the expected true positive (insider misuse detected) and false positive ("false alarms") rates? What does the "tradeoff curve" between true and false positives look like?

- What is the performance of Bayesian networks, neural networks, and decision trees on this task of learning predictive models from collected statistics?
- What is the computational burden of using gathered statistics to monitor for insider misuse?

We are currently undertaking an extensive empirical study to scientifically answer these questions, using real users (ten employees at Shavlik Technologies) in the Windows 2000 environment and the well-established experimental methodology of the machine-learning community.

For each measured system property, we are developing algorithms that choose the three best other measurements with which to *condition* the probability distribution for the given measurement (e.g., CPU cycles used by one's web browser as a function of overall CPU load, one's typing speed, and the time of day). We wish to also determine which *subsets* of all the measurements are the most effective. It is likely that having too many measurements will make it harder to accurately find the "insider intrusion" signal. In addition, reducing the number of system properties measured will lower the CPU demands of the intrusion-detection system. Finally, we aim to determine the best method for combining ("fusing") all the individual measurements into one composite score, whose value represents the likelihood of an intrusion.

Some of the Windows 2000 Properties We Measure

Our existing program collects real-time information from the following Windows 2000 sources:

- Performance Monitor (*Perfmon*) data,
- Event Log monitoring, and
- User and computer state information, such as typing rates, network traffic levels, programs running, and specific system API's invoked.

Current commercial IDS tools (e. g., from ISS and Kane) monitor either IP packets or log files, or some combination. Both provide useful information and are key to any IDS system. Our approach is unique in that it does not depend solely on network traffic levels, which can be inaccurate during busy times, nor does it depend solely on Event Logs, which can have a time lag and which do not contain all data-critical IDS information. For example, audit logs may have data that is too old to do real-time detection. (And system administrators may turn off event logging to save on performance or disk space without realizing they are severely hindering IDS systems.) Instead, we focus on effectively utilizing the rich sources of information made available by an advanced operating system such as Windows 2000, a topic for which we have some prior experience in a different context [GOECKS2000].

The Windows 2000 *Perfmon* utility contains a number of very useful security items that are often overlooked. These items can be accessed quickly. Categories that can be looked at include Network Performance, Disk Activity, Process Performance, Kernel Usage, and many others.

Event Log monitoring is done on key NT Registry locations, key system files, login abnormalities, and suspect account changes. In addition, invalid accesses to key files and to registry entries are monitored.

Using Window 2000's "hooks" facility, we also intercept keystrokes [MONROSE97] and mouse events. This allows our IDS to gather statistics on typical usage patterns by each desktop-computer's user. We collect statistics on such things as typing speed; usage (if any) of the numeric keypad; usage of function, cursor, and control keys; and speed of mouse movement. We hypothesize that all of this keystroke and mouse-usage information allows us to more accurately recognize cases where someone other than the "owner" of a desktop computer might be using it, for example if a secretary whose computer is in an open area leaves for lunch or the day without securing his or her computer.

Other items watched include the running of Windows 2000 Services and common programs, such as web browsers, Microsoft Office, editors, and program development tools.

Lastly, we monitor the calling of kernel security functions such as those that list the current user accounts. The calling of such functions is it a highly suspicious activity as it is common for hackers to do this as part of their attacks.

Some Preliminary Results

Our initial efforts have focused on the data concerning user keystrokes. We briefly report some preliminary results here; a future report will more precisely and completely describe our experimental studies. In our first set of experiments we are using ten experimental subjects, who are all employees of Shavlik Technologies performing their normal everyday activities. We have implemented an experimental approach that works as follows:

1. Use 50,000 keystrokes for each user to *train* a separate statistical model for each user. (On average, users type 5000 keystrokes per working day, so this is about two weeks of activity.)

2. Use another 50,000 keystrokes to *tune* some parameters (explained further below), separately for each user. The tuning data is collected from non-overlapping days than those used to create the training data. (Similarly, the testing data described below also comes from days separate from those where the training and tuning data were collected.)

3. Use a third set of 30,000 keystrokes as a *test* set. It is important for proper experimental methodology that one uses *separate* data to tune parameter and to estimate future accuracy. "Tuning on the test set" will usually lead to overestimates of future accuracy.

   For each person's test set, we simulate this person typing on all ten machines. We measure the fraction of times this typing is viewed as *not* coming from the owner of this machine (i.e., an *intrusion* was flagged). A flagged intrusion when a person is typing on their *own* machine is called a *false positive* (i.e., a false alarm). Conversely, a flagged intrusion when person *X* is typing on person *Y*'s machine is called a *true positive* (i.e, a correct alarm). Our goal is to have a high correct-alarm rate and an *very* low false-alarm rate, since system administrators will soon ignore an intrusion-detection system that generates too many false alarms.

The basic algorithm we are currently investigating works as follows:

1. Compute the probability of the last three keystrokes (including the time taken between keystrokes and the time each key was held down before being released), given the model learned for the given machine's normal user.
2. If this probability is lower than some threshold, *T*, then "mark" this keystroke.
3. If there are more than *N* marks in the last *W* keystrokes, raise an alarm.

Our algorithm automatically chooses the *T* and *N* settings for each person and for each *W*, based on optimizing accuracy on the *tuning* data set that was mentioned above, while holding the rate of false alarms to less than one per work day (actually, less than one per 5000 keystrokes, since that is the average number of keystrokes our users typed per day).

Our current accuracy results on the *test* data set, as a function of *W,* appear below. When measuring accuracies on the testing data, we use non-overlapping windows (of *W* keystrokes) in order to reduce the correlation between successive samples.

Window Width (*W*)	**Percentage of Intrusions Detected with a False Alarm Rate of Less Than 1 Per Work Day**
10	17.3%
20	30.4
40	53.9
80	71.2
160	86.4
320	94.6
640	97.4

We are encouraged that we can recognize a sizable fraction of intrusions (defined as user *X* typing on user *Y*'s computer) which such a low false-alarm rate, especially since we are currently only using in our experiments one type (i.e., keystrokes) of the many types of data we are collecting. Of course an intruder can do a lot of damage in, say, 160 keystrokes, but we believe that detecting with large windows can still be useful; it certainly is better than looking at yesterday's log files.

Some Related Work

Previous empirical studies have investigated the value of creating intrusion-detection systems by monitoring properties of computer systems, an idea that goes back at least 20 years [ANDERSON80]. However, prior work has focused on Unix systems, whereas over 90% of the world's computers run some variant of Microsoft Windows. In addition, prior studies have not looked at as large a collection of system measurements as we are using. For example, Warrender et al. (1999), Ghosh et al. (1999), and Lane and Brodley (1998) only look at Unix system calls, whereas Lee et al. (1999) only look at audit data, mainly from the TCP program.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.