

# User Profiling for Intrusion Detection in Windows NT

Tom Goldring

National Security Agency, 9800 Savage Road, Fort Meade, MD, 20755

## Abstract

In User Profiling, we observe the normal behavior of computer users and from this, seek to automatically learn models that characterize this behavior. Then for a new session, these models are used to either authenticate the login name, or to identify a malicious insider. A related problem is Program Profiling, in which models for normal activity of an application program are learned, then used to identify attacks. This is a somewhat easier problem because humans do not come with "specs", so compared to programs, our behavior is infinitely less predictable. In fact, a certain level of anomalous activity in human behavior is inevitable and must be taken into account.

Most if not all published work on this subject has used command line activity as its data source, collected on a Unix system. In this environment there are multiple ways to do most things, leaving much room for individual expression, yet even so the reported results have been less than stellar. Now consider today's point and click world, where command line activity is virtually nonexistent. Even worse, the Windows suite of interlinked applications provides a "path of least resistance", with the result that people look more alike than ever. Add to this the fact that much of the activity occurring on a host, especially if it's networked, is generated by the operating system and not user related. This requires massive filtering, but how to it accurately can be far from obvious. These considerations underscore the inherent difficulty of the problem.

For nearly two years we have been monitoring real users doing their daily work on an operational Windows NT network. This talk will describe the data we collect and methods we have used to analyze it, and present results obtained to date.

## 1 Introduction

Define *User Profiling* as observing what someone is doing on a computer, modeling this activity for the purpose of representing the user's "normal" behavior, and scoring new activity against the model. One reason for doing this would be to *authenticate* the user's login (verify the user is indeed who he claims to be). If we can capture a person's ingrained habit patterns in a model, hopefully this will provide a form of authentication that would be very hard to impersonate. It should come as no surprise that this turns out to be a highly nontrivial problem, because how to model human behavior is far from obvious. Even more difficult is to try and determine whether a legitimate user is doing anything malevolent (the *insider threat*). Hopefully such activity would stand out as anomalous when compared to the user's routine behavior, but to date, this hypothesis has not been verified experimentally.

A related discipline is *Program Profiling* e.g. as in [1], in which the normal behavior for an application program is modeled, usually for the purpose of detecting whether the program is doing anything it was not designed to do. We believe this to be a somewhat

easier problem because unlike humans, programs come with specifications and their behavior is therefore very limited by comparison.

## 2 Data

Let's consider some possibilities. Perhaps the simplest data source is the user's command line, e.g. [2, 3]. Such data has the advantage of being human readable, however it misses window behavior, and worse yet, it does not include commands executed inside a script, which is how many attacks are launched. Furthermore, in today's point and click environment this type of data is increasingly rare. Most if not all of the published literature on user profiling employs command line data, and while it may have been a viable data source years ago, we no longer believe this to be the case.

Program profiling typically employs *system call* data. For example, one can store a reasonably inclusive subset of the set of short sequences of system calls that appear during normal operation of the program, then define an *anomaly* as anything not in that sequence. More complicated methods have been studied but to our knowledge, these perform no better than the simple idea just described [4]. This data has very fine granularity and is appropriate for the purpose because for one thing, an application program is designed to fulfill a specific function and the set of system calls it invokes is therefore limited. Secondly, one only looks at those system calls generated by the profiled application. Unfortunately, neither of these conditions applies to user profiling. Here one would have to look at *all* the system calls, most of which are generated by the operating system in the normal course of doing what it needs to do and are not user related. Such data is simply noise to us and is very difficult if not impossible to filter out. Another disadvantage is that system call data is not human readable. We consider readability important because research on user profiling is in its infancy, therefore we need to understand the data before we can hope to model it effectively. For these reasons, we feel that system call data is inappropriate for user profiling.

We prefer two other data sources, one of which is the *process table*. A "process" is defined as an "instance of a program in execution" [5], and the process table is basically the bookkeeping mechanism that a multitasking operating system uses to share *cpu* (central processing unit) time among the various things that are running concurrently. We can sample it two or three times per second, recording three types of events: birth (the process is there now but wasn't there the last time we looked), death (it was there the last time but isn't anymore), and continuation (an existing process uses some cpu time). We believe that this data has all of the advantages and none of the disadvantages we have spoken of:

- one can read it and guess what the user is doing
- it includes window behavior, and anything running in a script
- it is a necessary component of any multitasking operating system and isn't about to go away
- the level of granularity is somewhere in between the levels for command line and system calls respectively
- most if not all of the system noise can be filtered out, as we show in the next paragraph.

Additionally, process data has a natural tree structure, in that every process (other than the "initialization" process started by the operating system at boot time) is launched by some other process and therefore has a unique parent.

If process data is augmented with a second data source, found on any windowing system, we can filter out system noise by exploiting the tree structure. We know that anything the

user does is always done in some window, and each window has a title bar containing some text describing it. Our second data source is this set of window titles, and a record is generated each time a new window comes up, or the contents in an existing window's title bar change. We also save the window's process and parent process ID's. Now for any of the three types of events resulting from sampling the process table, we check to see whether it is descended from some existing window, or its parent or grandparent. If not, we consider it to be system noise and throw it out. In this way we obtain human readable data that faithfully represents the user's behavior to a high degree of accuracy.

A sample fragment of the post-processed data appears in Slide 9 of the accompanying Powerpoint presentation. Column 2 shows the elapsed time in seconds since login. Reading this data shows that the user was reading email, then switched to Windows Explorer and double clicked on the file "things to do.dat", which opened Wordpad as the default application.

Finally, window titles allow us to solve the "explorer" problem: in the Windows operating system, the process for the file browser is called Explorer, but so is the operating system itself and many of the built in programs, such as Control Panel and Find Files. These would not be distinguishable from the process table alone. Additionally, window titles contain much information of interest to us, such as email subject lines, and names of documents, web pages, and file folders.

### 3 Feature Selection

A number of existing classification methods could be used to learn models from our data and classify new user sessions (a *session* is defined as everything the user does from login to logout), provided we first convert the data into an appropriate form. Typically, these methods require data matrices, where each row of the matrix is a separate observation and each column is some attribute or *feature* belonging to that observation. For example, in demographic data one might have a row for each person sampled, and the features might be things like, height, weight, gender, age, etc. However for our data, feature selection is somewhat less obvious, partly because we are mixing two different data types (windows vs. process table). Ideally, feature values should be different for different users, but similar for different sessions belonging to one user.

In Slide 10 of the Powerpoint, we show session plots for three different users (one per row), five sessions per user. The  $x$  axis represents time (login to logout read left to right), while the process names have been mapped to points on the  $y$  axis. So each plot shows the processes the user invoked during the session, color coded by *cpu* time (black, red, orange, yellow, green blue, pink, and white, in that order, range from min to max for that row). By visually following horizontal lines across the five plots in each row, we see that there is indeed a certain degree of similarity among the processes a user invokes from one session to another. Slide 11 shows the same 15 plots, with each one rotated 90° clockwise. One can see that trying to follow a line vertically from a session to the one underneath (which belongs to a different user) is generally unsuccessful. Note also the last two sessions for the third user are quite anomalous compared to the others, illustrating the fact that it's perfectly normal for a person to do abnormal things once in a while.

In Slide 14, we show probability density functions (one for each of five users) for some candidate features:

1. time between windows
2. time between new windows
3. number of windows open at once

4. number of windows open at once, weighted by amount of time open
5. number of words in window title
6.  $\frac{\text{\# of words in window title belonging to Windows}}{\text{total \# of words in window title}}$

Regarding feature 6, examples of “words belonging to Windows” in the sample data fragment (Slide 9) are Inbox, Microsoft, Outlook, and Wordpad. One can see that features 2, 4, and 6 do better at distinguishing among users than do their simpler counterparts 1, 3, and 5.

After experimenting with a number of numeric feature candidates, we selected eight of these, and generated the corresponding session plots for the same 15 sessions seen previously (Slide 15). Here time is on the  $y$  axis (start of session = top, end = bottom). Column 1 denotes relative time within the session, while the remaining columns correspond to the eight features. Due to their nature, features are generated at different times, so we are looking at *data image* plots of numeric matrices (one per session) where each row is a vector with time as its first component, and each other component is nonzero only if the corresponding feature was generated at that time. The last three features are derived from window titles, therefore these always appear concurrently. So for example, we can see black areas where the user was running some application but did not interact with the computer directly, as there was no window activity.

As before, sessions for any given user should look similar, but different from those of other users. This appears to be the case if we look at plots of the first two principal components (Slide 16).

## 4 Future Work

In the very near future we will test some classification algorithms against our data (to date, 26 users and 1292 sessions total). For example, we might try Naive Bayes, Random Forests, and Support Vector Machines. Farther out in time, we may choose to investigate the following:

- Build in specific methods for major application programs
- Monitor keystrokes and mouse movements
- Characterize insider misuse.

## References

- [1] S. Forrest, S. Hofmeyer, and A. Somayaji, “Computer Immunology”, *Comm. of the ACM*, vol. 40, no. 10, pp. 88-96 (1997)
- [2] T. Lane and C. E. Brodley, “Sequence Matching and Learning in Anomaly Detection for Computer Security”, *AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, Providence, RI (1997)
- [3] M. Theus and M. Schonlau, “Intrusion Detection Based on Structural Zeroes”, *Stat. Comp. & Stat. Graph. Newsletter*, vol. p no. 1, pp. 12-17 (1998)
- [4] S. Forrest, private communication (2002)
- [5] M. J. Bach, *The Design of the Unix Operating System*, Prentice-Hall (1986)