User Modelling for Exclusion and Anomaly Detection: a Behavioural Intrusion Detection System

Grant Pannell and Helen Ashman

[1] WebTech and Security Lab, University of South Australia,
{grant.pannell | helen.ashman}@unisa.edu.au

Abstract. User models are generally created to personalise information or share user experiences among like-minded individuals. An individual's characteristics are compared to those of some canonical user type, and the user included in various user groups accordingly. Those user groups might be defined according to academic ability or recreational interests, but the aim is to include the user in relevant groups where appropriate. The user model described here operates on the principle of exclusion, not inclusion, and its purpose is to detect atypical behaviour, seeing if a user falls outside a category, rather than inside one. That is, it performs anomaly detection against either an individual user model or a typical user model. Such a principle can be usefully applied in many ways, such as early detection of illness, or discovering students with learning issues. In this paper, we apply the anomaly detection principle to the detection of intruders on a computer system masquerading as real users, by comparing the behaviour of the intruder with the expected behaviour of the user as characterised by their user model. This behaviour is captured in characteristics such as typing habits, Web page usage and application usage. An experimental intrusion detection system (IDS) was built with user models reflecting these characteristics, and it was found that comparison with a small number of key characteristics from a user model can very quickly detect anomalies and thus identify an intruder.

Key words: user model, exclusion, anomaly detection, behavioural IDS

1 Introduction

1.1 How User Models are Applied

User modelling is frequently a part of applications where some element of personalisation is required in information delivery. Commercial applications such as recommender systems and educational delivery systems are two areas where user models are now a mainstream technology. In particular, adaptive hypermedia systems have focused on delivery of educational materials for many years [4], but there is now a greater awareness of the potential application areas of user models, such as Cultural heritage, Health care, Assistive technologies and so on.

Interestingly, these application areas generally implement user models with the intention of being able to classify a user into a particular group, with the hope that the aggregated experiences of other members will be able to assist that user. The purpose

of the user modelling is to *include* the user into some group of like-minded others. In recommender systems in particular, the user model allows the inclusion of an individual into a larger group based on their resemblance to the majority of the group.

The approach in this paper departs significantly from this. We propose an alternative use of user models which is much less common in the literature, namely to model a user with the intention of determining if they do not belong in some group, namely, the aim is to *exclude* users or at least to identify non-members. In many cases, a user is assumed by default to belong to a certain group, such as the group of permitted users, or the group of healthy users. Where such implicit judgements are made about users, it can be helpful to identify when the assumption is wrong. Examples include identifying where users are no longer part of a group of healthy people (i.e. identifying illness or stress), or identifying when students do not belong in a student group (i.e. a person masquerading as another student in order to achieve better results by proxy). In particular, the work reported here is motivated by a need to exclude unauthorised users who may have bypassed the barrier security methods (such as passwords) and have entered a computer system.

We propose the use of anomaly detection over user models, where anomalies indicate that the user is behaving in an uncharacteristic way. This might be a user who normally opens word-processing files now creating Unix executables, or someone with error-prone typing suddenly showing proficiency. To perform anomaly detection successfully, it is necessary to create a user model that reflects characteristics of the individual, and to perform statistical analyses that identify anomalies.

This work is similar to and extends the user-recognition work that statistically analysed Unix command line data in order to characterise and subsequently recognise an individual [9]. Anomaly detection is a natural use of such automatically-recognised users and Unix command line data has been successfully used for around 15 years in intrusion detection systems, although the data being analysed is not purely user commands but also includes keystrokes, session times and lengths and resource usage (see Sect. 1.2). This paper also extends that of prior IDS research on user-specific anomaly detection, as it performs analyses over multiple characteristics, not just command line data, but also over GUI-based characteristics such as Web page accesses and game playing. We also consider different classes of users, categorised according to their proficiency (basic computing skills versus deep technical knowledge), purpose of use (work versus leisure) and direct versus indirect user characteristics (inferred data such as typical CPU and memory use versus direct user data such as keystrokes). Interestingly, we found that the most useful characteristics for anomaly detection (and hence for user recognition more generally) were user-specific ones that directly reflect some user feature, such as typing habits and Web page usage, more so than application-specific features which only indirectly reflect user activity.

Another aspect of this work is that instead of comparing the user's current behaviour against a stereotypical user model or aggregated user model, we compare a user's model against their own past behaviour. That is, the user model is compared over time, rather than over a user group. This alternative approach is starting to appear in medical uses, such as detecting degradation in a user's skills over time [17].

1.2 Overview of Intrusion Detection Systems

The main focus of this paper is on the application of user models to anomaly detection. However since the principle is demonstrated in an intrusion detection system (IDS), we briefly overview intrusion detection systems here, focusing mostly on IDSs which capture and use information about users.

There are two main principles behind intrusion detection systems, the first being *rule-based systems*, where rules explicitly describe disallowed activities (e.g. use of the "su" command, or any access within a honeypot area), and *anomaly-based systems*, where specific behaviours are not prescribed but activity significantly outside the norm triggers an alert. These latter are generally statistical systems where a range of acceptable usages is characterised, either by statistical analysis of typical activity, or by defining a "canonical" activity, such as the number of accesses from a given IP in a time frame, or the number of some types of accesses (e.g. "ping" commands).

Intrusion detection based on user models is very much at an experimental stage, but generally can be characterised as being a sort of statistical, anomaly-based IDS except that the statistics are not calculated on the network traffic but instead are calculated on user behaviour. This approach implements a *canonical* user model that represents the trusted user (either a typical user or an identified individual), and a *sessional* user model, representing the activity and behaviour of the user who is currently operating under the appropriate user ID, which is compared to the trusted model. Should there be a significant discrepancy between the canonical user model and the sessional user model, an alert is generated. The canonical user model is of course persistent, while the sessional user model is only sessional (although pertinent data is retained for evidence or for statistical analysis).

In intrusion detection, user models can be applied in different ways:

- *"Role"-based canonical modelling*: An IDS may not map a user's activities to a specific individual's user model, but rather compare the sessional user model against a "typical" user model. If a user's activity falls outside the "typical" behaviour, an alert is generated [15]. However this is not especially useful in an environment where there are no "typical" users or working behaviours.
- *Personal user models*: We aim to detect intrusions by comparing the behaviour of a user currently active with their own user model of past behaviour. It will compare the user's behaviour in such things as common typing errors, frequently-used commands and applications, normal times and duration of connections, command sequences, and so on. This is the application area described in this paper that demonstrates the usefulness of anomaly detection over user models.

Anderson [1] outlines the idea of audit trails, where activity logs are taken from machines and manually analysed by activity security officers to find unauthorized access to files and other resources. He introduces the idea of using an automated surveillance system that looks out for characteristics such as session logs, durations, program usage, device usage, and file usage. The proposed surveillance system used basic statistics, such as averages, standard deviations, maxima and minima, to determine abnormal usage of the machine.

Denning [5] furthers Anderson's work by formally defining the structure of an IDS that is capable of automated audit trail analysis and adds other characteristics such as

CPU and memory usage. This focuses on the IDS being statistical rather than rule-based. Lunt [11] extends Denning's structure by combining the rule-based system with statistical methods. One of the first implementations of a behavioural intrusion detection system, based on Denning's work, analysed daily audit files to find anomalous activity against user behaviour models and specific constraints [15].

User profiling for intrusion detection is established in the Unix systems and many have profiled users with command line data [2,8,16]. Characteristics used include commands, session times and resource usage, and the data analysis algorithm.

Very few user-profiling IDSs have combined the characteristics to improve performance although it is done quite often in network IDSs and it has been shown that the performance of a network IDS can be increased by using multiple, differently implemented, network IDSs [10]. Other experiments with using OR and AND operations with multiple systems improve both detection and false-positive rates [7]. One of the contributions of this paper is to implement a behavioural IDS that combines numerous characteristics in the same way that network IDSs do.

Adapting the IDS to a GUI-based system influences the user characteristics that are modelled. For example, capturing data from the mouse and from the way the user interacts with their windows now becomes possible. Implementations of a behavioural system built on GUI-based systems seem to be rare. One example bases the system on Windows 2000, using data from the operating system's performance monitor to create user profiles, and it is claimed that this system obtains a 95% detection rate and a low false-positive rate of "less than one alert per day" [14].

The physical attributes of the user can be characterised such as mouse movements [13] and the delays between keys typed on a keyboard [3]. It is feasible to use key delays and typing patterns to determine whether the user is cognitively or physically stressed [17]. This application clearly shows the potential benefits of anomaly detection over user models within health applications.

There have also been other profiling IDSs, which however are not user-focused, such as profiling network traffic from a host to determine whether it has been compromised [12] and profiling the system event log to determine the execution flow of an application to ensure that it is not being misused or exploited [6].

In summary, behavioural intrusion detection systems have so far seen little work in the combination of user characteristics, especially where data can be collected from GUI-based sources. The work reported in this paper considers intrusion detection applied over combined user characteristics and for individual users of many different types, such as office worker, advanced worker, game-playing and so on.

2 The User Model and the IDS

2.1 Characteristics Stored in the User Model

The system profiles a user using multiple characteristics, and combines different statistical methods such as standard deviations, averages, and limits, and then uses a score-based system to determine whether the combination of characteristics triggers

an intrusion based on previous user data. The profiling engine retrieves characteristics at an interval of 30 seconds to ensure that new anomalies are detected quickly.

The system profiles users around the following characteristics:

- Applications running - The applications that are running on a machine would allow profiling of a user to determine the default or typical applications they use. For example, a user may exclusively use a specific set of office applications. Any new applications opened by the user could indicate anomalistic behaviour;
- Number of windows open - The number of windows currently open also can determine one user from another, depending on their style of use;
- Performance of running applications - Performance of running applications, such as measured by CPU usage and memory usage could determine how the applications are being used. For example, an abnormally high CPU usage in a database application could mean an intruder is extracting data;
- Keystroke analysis - Keystroke analysis includes such characteristics as speed, combination of keys and pauses between key presses; and
- Websites viewed - The websites viewed characteristic looks at web browser history to determine if new sites have entered the profile.

2.2 Analytical Algorithms Performed over User Characteristics

The *CPU usage* and *memory usage* characteristic engines were a collection of three different algorithms and a scoring system. The algorithms used included a Standard deviation analysis, a Rolling average, and Upper and lower limits. The standard deviation analysis stored the previous 120 values collected by the system, i.e. the past 60 minutes of data with an interval of 30-second collections. An anomaly was detected if a new value was three deviations from the mean. The rolling average was simply a cumulative average that took the previous usage value, added it to the new value, and divided it by two. This gives an overall view of the process for its entire lifetime, compared to the standard deviation algorithm, which only provides a view over the past hour. The final algorithm used, sliding limit, would only be changed during the learning phase of the system. This allows the system to learn the upper and lower limits of the process, and since it is assumed that the profile is perfectly trained, these values should show when the process is acting abnormally.

After these three algorithms had analysed the incoming data, a scoring system was used to validate any detected anomaly. If more than one algorithm triggered, or if the same algorithm triggered multiple times, it is less likely to be a false-positive. The scoring system essentially attempts to smooth out false-positives given by the simple algorithms. Each algorithm was assigned a point value: Standard Deviation given 0.5 points, Rolling Average getting 1 point and Sliding Limit getting 2 points. The points would accumulate over three collections and once the 3-point limit was breached, the system would trigger an anomaly, otherwise the accumulated points would reset to 0.

The *number of processes* characteristic looked at two different sets of user data related to number of processes, namely the variance of number of processes, over the hour and the number of new processes to the user's profile. The variance of number of processes per hour was calculated the same as the standard deviation algorithm used for CPU usage and memory usage characteristics. The number of processes are

collected every 30 seconds over the past hour and the mean and standard deviation are calculated. If the current number of processes were 2.5 deviations away from the mean, the number of running processes would be anomalistic. The number of processes characteristic also had to determine if the process was new to the profile, and if so, the system should theoretically mark as anomalous all new processes. In a real-world situation this may be desirable. However, it is dependent on the actual user type, e.g. a power user frequently installs new applications.

The *number of windows* characteristic algorithm works exactly like the number of processes algorithm except that only the variance of the number of windows per hour is used and the system does not look for new windows to add to the profile.

The *websites viewed* characteristic algorithm also works in this way but looks at the number of new sites that had been added to the browser's history, per hour.

Finally, the *keystroke analysis* algorithm adapted the ideas from Bergadano et al. [3], using digraphs to capture a user's typing pattern. If a user were to type "Digraphs", the system would store key pairs: "Di", "ig", "gr", "ra", "ap", "ph" and "hs" with the delays between each of the keys. To determine if a typed digraph was anomalistic, the algorithm would determine the standard deviation and mean then check if the new delay value was more than 2.5 standard deviations from the mean.

2.3 System Architecture

The architecture is based upon a generic intrusion detection system. The data collection engine gathers data from the performance monitor, Windows API, browser history locations and from keystrokes, at an interval of every 30 seconds. It then sends data for each user characteristic to its own analysis engine. The analysis engine contains two types of detection: rule-based and anomaly. Both types analyse the data to determine if the behaviour falls within the normal range of the user's profile, according to the algorithms described above (see Sect. 2.2). If any anomaly is significant, the user's activity for that characteristic in that 30-second time slice is deemed to be unauthorised.

Each characteristic engine then sends its results to the data-mining engine which collects results and determines what action to take. The engine uses a score-based threshold system, determined during testing, and past results to lower false-positives. This may mean that multiple alarms will need to be triggered or the same alarm triggered many times for the system to take action. The data-mining engine then sends its results to the alert/action engine that will perform actions specified by the user depending on if the system has determined authorised or unauthorised behaviour.

3 Design of Experiment Validating Behavioural Anomaly Detection

Eleven users tested the system, running the IDS in the background for approximately 10 days. The systems that were used had a range of different operating systems and uses. Each user categorised their machine for its primary use as follows:

No.	Machine main use
3	Web Browsing - Primarily surfing the web
1	Gaming - Primarily playing 3D games
3	Office/University Work - Primarily using Microsoft Office-like tools
0	Entertainment - Primarily using a media player to watch or listen to media
4	Power User - A combination of the above plus arbitrary other applications

Data was gathered in two modes: learning, and intrusion detection. In learning mode, the system treated all user actions as normal and added them to the profile, i.e. the learning phase was assumed to be clean of intrusions. However while in learning mode, especially initially, most of the user actions were anomalous compared to the user model which was necessarily incomplete at this stage, so the reduction in anomalies (i.e. false-positives) detected was used as a measure of how successfully the system was able to create the user's profile of normal behaviour, especially for each profiled characteristic. The system was placed in learning mode for 10 days (28880 30-second collections), after which it was switched to detection mode, where the system treated anomalies as intrusions and would create an appropriate alert.

The experiment then aimed to generate anomalies against the user profile by stressing each characteristic in two ways, firstly by deliberately challenging each specific characteristic such as by changing a keystroke pattern, visiting unusual websites or running unusually CPU-intensive processes. The second testing method was done by putting a new user in front of the machine, effectively masquerading as the legitimate user. This allowed us to measure how long the system took to detect a different user and what characteristics were best for this.

4 Results and Discussion

4.1 In the Learning Phase – Separate Characteristics

In this section, we consider how frequently anomalies were detected for each user characteristic, focusing on those reflecting the user's behaviour directly, and then how the data mining engine that combines them all performed during this phase.

The assumption was made that during the learning phase, the machines were not exposed to intrusion and that all user activity was valid. However it was useful to generate a measure of how well and how quickly the system learned the user's typical behaviours. We measured to what extent the user profile had converged on a typical collection of values for the given characteristics by counting the number of "false-positives" during the learning period, i.e. the number of times some user behaviour appeared to be anomalous, according to the as-yet-incomplete user model. Because the learning phase was not exposed to intruders, the apparent anomalies were not actually anomalies but were known to be false-positives. The number of false-positives fell as the user model became more representative of normal user activity.

All characteristics showed a steadily-decreasing learning curve, where the number of false-positives would decrease over time. This shows that the IDS was successfully

able to learn the user's activities during the learning period. In fact, for some classes of user it would be feasible to reduce the learning time since the number of false-positives later in the learning period became negligible and the system was not learning anything "new" by continuing to observe the user.

The *CPU usage characteristic*, overall produced a significant number of false-positives compared to other characteristics. Nevertheless, the system showed a learning curve that resulted in fewer than 50 false-positives per two days near the end of the learning period. The power user machines produced the highest false-positive rate of 6.73% during the learning period, showing that power users have a wider range of typical usages and that perhaps ten days is insufficient for learning the typical behaviour of a power user. The lowest CPU usage false-positive rates during the learning phase come from the web browsing and office machines, ranging between 0.6% and 0.8%, indicating a lower level of variance in such activity. The CPU usage characteristic may not be the most suitable for anomaly detection, given its ongoing false-positive rate and inapplicability across all user types. It also does not directly reflect user personal characteristics except very coarsely, such as when a user is imposing an abnormal load on the processor.

The *memory usage characteristic* produced fewer false-positives per day during the learning period, but also showed a shallower learning curve. This suggests that the system was not as successful in profiling the memory usage of processes on the machine. By the end of the learning period, the memory usage characteristic would output approximately 30 false-positives per two days. Again, the highest false-positive rates come from the power user machines at 1.18%. Like the CPU usage characteristic, this characteristic may be less useful for anomaly detection. It also does not directly reflect the user's personal characteristics except very coarsely, and may be prone to error when very large files are opened.

The *websites viewed* and *number of windows* characteristics were more promising. They all produced the fewest false-positives with rates of less than 0.05%. While these characteristics were not triggered as often, they show a steady decrease over learning period. The false-positives for the websites viewed characteristic would occasionally increase over time then continue to decrease. This might be explained by the user initially viewing a usual set of sites, and then changing their pattern by browsing to new sites. The system increases its alert level during the pattern change over and then decreases as it learns the new behaviour.

The characteristic that provided the most rapid learning, and perhaps the one most promising for profiling a user, is *keystroke usage*. This is likely due to the physical relation it has to the user. Again, the highest false-positive rates come from the power user machines at approximately 1.89% although the system would output fewer than 50 alerts per two days at the end of the learning period.

The gaming machine, in keystroke usage results, has a high false-positive rate of 3.84% with false-positives equally spread over collection periods, meaning the system could not determine a consistent typing pattern. This may be because when gaming, the user is reacting to what they see on the screen and the delay between keys depends more on the game, i.e. user activity is not autonomous but rather interactive. In contrast, an office machine user tends to generate a more consistent typing pattern.

Aside from the gaming machine, the keystroke analysis was one of the more successful characteristics for profiling quickly with the majority of users shown to

have an easily-characterised typing style. The false-positive results for the keystroke usage characteristic for the different user classes is summarised in Fig. 1.

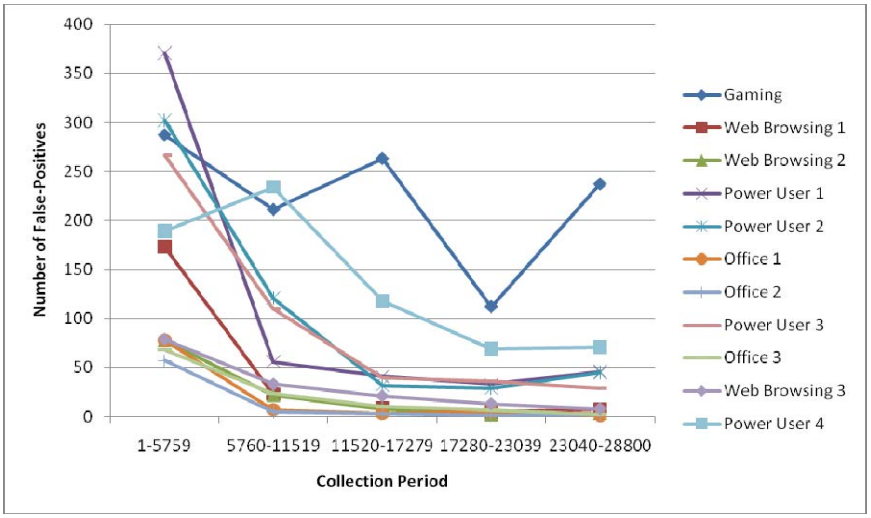

Fig. 1. False-positive rates over the learning period for keystroke usage

4.2 In the Learning Phase – Combined Characteristics

Finally, we look at the combination of outputs by the *data mining engine* using a scoring system which combines the anomalies detected by the separate characteristics.

Once the scoring system is applied to combine the characteristics via the data mining engine, the learning curve starts to flatten and the system begins to output the same number of false-positives per day. This is not necessarily good as it is likely that the system will continue to output the same level of false-positives in the future. However, the flat curve shows a consistent false-positive rate and with improved algorithms, so this rate could possibly be decreased. The false-positive rates during learning mode peak at 0.62% for a power user machine, representing fewer than 10 false-positives per day. This indicates that a combination of characteristics is significantly better than any single characteristic. Figure 2 shows the false-positive rate during the learning period for combining characteristics using the scoring system.

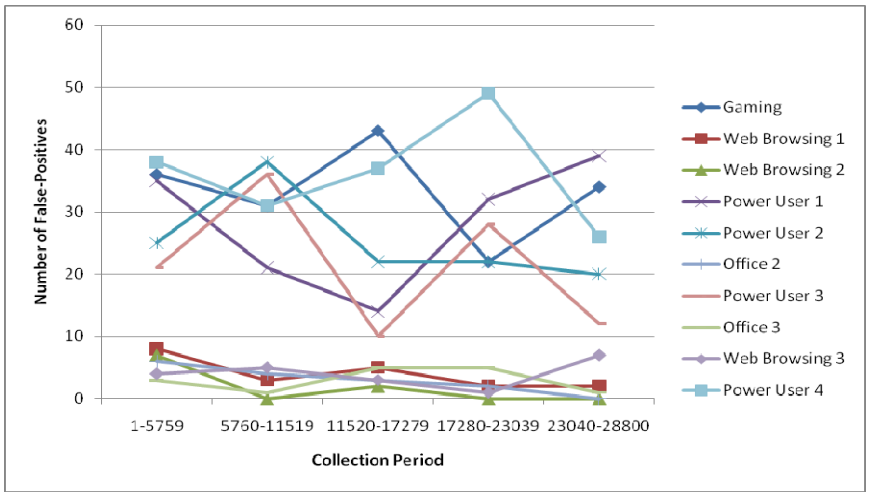

Fig. 2. False-positive rate during the learning period using scoring system

These results indicate that the system can learn typical user behaviour in a reasonable time by observing normal use during a dedicated learning. The most effective characteristics are those that directly reflect user behaviour, such as keystrokes, websites visited and number of windows opened. These are governed by the user's own physical attributes (keystrokes) or their own choices (opening windows or selecting websites to view) and are easier to characterise than those only indirectly related to the user, which may be affected by other causes other than the user.

In the scoring system, the number of false-positives was much lower than the false-positive rate for some of the individual characteristics, showing that the scoring system was able to correct for false-positives from individual characteristics.

4.3 Anomaly Detection Phase

Once the learning phase was completed, the system was switched to detection mode where instead of subsuming the detected new behaviour into the user model, the system triggers an alert when an intrusion occurs.

Anomaly detection was tested by stressing each characteristic with an abnormal usage, such as opening many windows. It was also tested by placing a new user in front of the machine and waiting for the system to trigger with their usage pattern.

The keystroke analysis engine triggered the system most quickly, generally in less than 120 seconds (4 cycles of data collection). The more application-related characteristics (number of processes, CPU usage and memory usage) all had higher detection times, with an average of 180 seconds. Placing a new user in front of the machine triggered alerts in the shortest time (90 seconds), for almost all machine usage types. Figure 3 shows the average time taken to detect intrusions on all machine usage types.

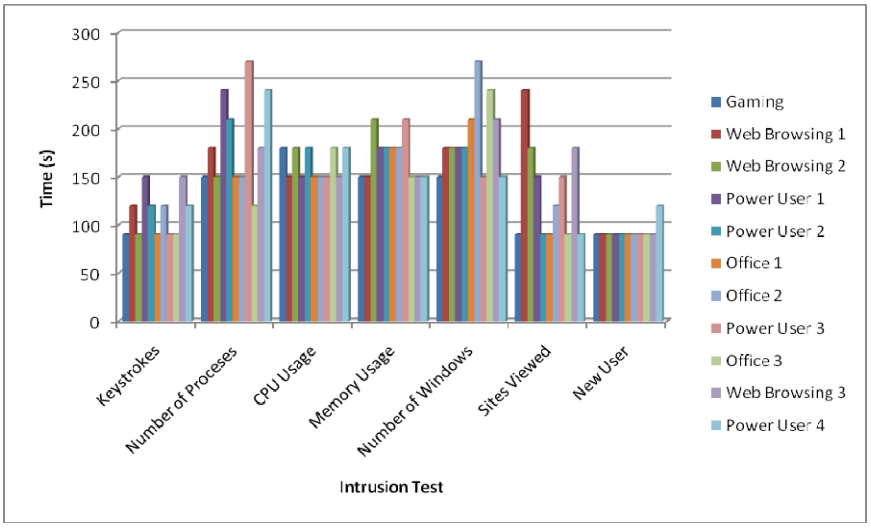

Fig. 3. Average time to detect intrusions on all machine usage types

The most useful characteristics are those that directly reflect user behaviour, keystrokes and sites viewed. The number of windows characteristic, also user-specific, is the slowest,. The less direct characteristics were the least efficient for identifying anomalies, requiring up to 10 cycles of data collection.

The ability to detect genuine intrusions within three data collections shows that a combination of characteristics is more effective than any individual characteristic, even when the scoring system that combines the characteristics is not optimised.

5 Conclusions and Future Work

This paper reports on the facility of user models for anomaly detection, and that a combination of user characteristics achieves the most rapid detection. The more personal characteristics were the most efficient for anomaly detection. However there are still many improvements possible, such as more personal features in the user model. This could include mouse use and n-gram analysis of writing styles for commonly-used words and phrases. It may also be possible to create a database to categorize websites and applications. e.g. to track the primary text editor used by the user. We aim to optimise the scoring system and learning algorithms to achieve detection in a single collection. More complex algorithms, such as a genetic algorithm, could increase the performance of the system. It may also be valuable to use anomaly detection over other user models to see if the user is behaving abnormally, e.g. fluctuations in student achievement may indicate a false user.

References

1. Anderson, J.: Computer Security Threat Monitoring and Surveillance. James P. Anderson Co., Fort Washington (1980)
2. Balajinath, B., Raghavan, S. V.: Intrusion detection through learning behavior model. In: Computer Communications, vol. 24, Issue 12, pp. 1202-1212. ScienceDirect, Amsterdam (2001)
3. Bergadano, F., Gunetti, D., Picardi, C.: Identity verification through dynamic keystroke analysis. In: Intelligent Data Analysis, vol. 7, Issue 5, pp. 469-496. IOS Press, Amsterdam (2003)
4. Brusilovsky, P.: Methods and techniques of Adaptive Hypermedia. In: User Modeling and User Adapted Interaction, vol. 6 num. 2-3, pp 87-129. Springer, Heidelberg (1995)
5. Denning, D. E.: An Intrusion-Detection Model. In: IEEE Transactions on Software Engineering, vol. 13, Issue 2, pp. 222-232. IEEE Press, Piscataway (1987)
6. Forrest, S., Hofmeyr, S. A., Somayaji, A., Longstaff, T. A.: A sense of self for Unix processes. In: Proc. 1996 IEEE Symposium on Security and Privacy, pp. 120-128. IEEE Computer Society, Washington (1996)
7. Gu, G., Cardenas, A. A., Lee, K.: Principled reasoning and practical applications of alert fusion in intrusion detection systems. In: Proc. ASIACCS '08, pp. 136-147. ACM, New York (2008)
8. Gunetti, D., Ruffo. G: Intrusion Detection through Behavioral Data. In: Proc. 3rd Intl. Symposium on Advances in Intelligent Data Analysis, LNCS, vol. 1642, pp. 383-394. Springer-Verlag, London (1999)
9. Iglesias, J.A., Ledezma, A., Sanchis, A.: Creating User Profiles From a Command-Line Interface: A Statistical Approach. In: Proc. UMAP 2009, pp 90-101. Springer-Verlag, Berlin (2009)
10. Julisch, K., Dacier, M.: Mining intrusion detection alarms for actionable knowledge. In: Proc. 8th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining, pp. 366-375. ACM, New York (2002)
11. Lunt, T. F.: Real-time intrusion detection. In: COMPCON Spring '89. 34th IEEE Computer Society Int. Conference: Intellectual Leverage, Digest of Papers, pp. 348-353. IEEE Press, Washington (1989)
12. Mazzariello, C., Oliviero, F.: An Autonomic Intrusion Detection System Based on Behavioral Network Engineering. In: Proc. INFOCOM 2006, pp. 1-2. IEEE Press, Washington (2006)
13. Pusara, M., Brodley, C. E.: User re-authentication via mouse movements. In: ACM workshop on Visualization and data mining for computer security, pp. 1-8. ACM, New York (2004)
14. Shavlik, J., Shavlik, M.: Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In: Proc. 10th ACM SIGKDD, pp. 276-285. ACM, New York (2004)
15. Smaha, S. E.: Haystack: an intrusion detection system. In: 4th ACSAC, pp. 37-44. IEEE Press, Washington (1988)
16. Tan, K.: The application of neural networks to UNIX computer security. In: IEEE International Conference on Neural Networks, 1995. Proc., vol. 1, pp. 476-481. IEEE Press, Washington (1995)
17. Vizer, L. M., Zhou, L., Sears, A.: Automated stress detection using keystroke and linguistic features: An exploratory study. In: IJHCS, vol. 67. Issue 10, pp. 870-886. Academic Press, Inc., Duluth (2009)