AFRL-IF-RS-TR-2006-263
Final Technical Report
August 2006

A COMPREHENSIVE REASONING FRAMEWORK FOR INFORMATION SURVIVABILITY (USER INTENT ENCAPSULATION AND REASONING ABOUT INTRUSION: IMPLEMENTATION AND PERFORMANCE ASSESSMENT)

State University of NY at Buffalo

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. Q010/00

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

STINFO FINAL REPORT

AFRL-IF-RS-TR-2006-263 has been reviewed and is approved for publication.

APPROVED: /s/

KEVIN A. KWIAT
Project Engineer

FOR THE DIRECTOR: /s/

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* AUG 2006	**2. REPORT TYPE** Final	**3. DATES COVERED** *(From - To)* Mar 00 – May 06

4. TITLE AND SUBTITLE A COMPREHENSIVE REASONING FRAMEWORK FOR INFORMATION SURVIVABILITY (USER INTENT ENCAPSULATION AND REASONING ABOUT INTRUSION: IMPLEMENTATION AND PERFORMANCE)	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER F30602-00-1-0507
	5c. PROGRAM ELEMENT NUMBER 62301E
6. AUTHOR(S) Shambhu Upadhyaya	**5d. PROJECT NUMBER** 2301
	5e. TASK NUMBER 04
	5f. WORK UNIT NUMBER 03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) State University of NY at Buffalo, Dept of Computer Science & Engineering, 201 Bell Hall, Buffalo NY 14260; Colorado State University, Dept of Computer Science & Engineering, 601 S. Howe St., Fort Collins CO 80523-1873	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA, 3701 N. Fairfax Dr., Arlington VA 22203-1714; AFRL/IFGA, 525 Brooks Rd, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2006-263

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 06-574

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This effort approaches the problem of user-level intrusion detection by investigating the design and implementation of a practical online user-level intrusion detection system. The outcome of this research is a Dynamic Reasoning based User Intent Driven (DRUID) intrusion detection system. It is important to pay attention to deployment-time issues such as usability and evasion, otherwise it may lead to a situation where the security system is deployed but is either unusable or is deliberately bypassed. A variation of sequential hypothesis testing is proposed to address these issues. Data plays a very important role in the validation of any new approaches or models that are proposed. Unfortunately, in the user-level intrusion detection domain, due to concerns of privacy, there are too few datasets available to the research community. This issue is addressed by devising a data generation algorithm called RACOON based on a model used to profile users.

15. SUBJECT TERMS
Intrusion Detection, Anomaly Detection, User-level Intrusion Detection, Information Survivability

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Kevin A. Kwiat
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UL	41	**19b. TELEPONE NUMBER (*Include area code*)**

Table of Contents

List of Figures

List of Tables

1 Summary

Intrusion detection attempts to detect attacker activity should the preventive measures be inadequate. Several research efforts have proposed schemes to perform user-level intrusion detection using statistical anomaly detection techniques. However, few practical implementations of anomaly detection systems are currently known. Major hindrances in this regard are poor accuracy of detection and false positives. While some of the reasons may be attributed to theory and technology, a major factor that is overlooked is the user. In this project, we developed a novel approach that brings the user into the loop by querying him for his session intent in a proactive manner. The owner intent so encapsulated serves the purpose of a certificate based on which more accurate intrusion detection decisions can be made. We approach the problem of user-level intrusion detection in a more holistic manner by investigating the issues regarding design and implementation of a practical online user-level intrusion detection system. The outcome of this research is a Dynamic Reasoning based User Intent Driven intrusion detection system called DRUID.

From the scalability point of view, a novel higher order representation of a user's profile is proposed for DRUID, which includes the hierarchical notion of jobs/tasks, followed by the basic units of functionality which a user requires to accomplish these tasks and then the actual commands. Such a representation is a significant departure from known techniques and provides several benefits such as user involvement in the security process, lowered false positive rates and per-job profiling. While new security approaches are constantly proposed, very few pay attention to deployment-time issues such as usability and evasion. These are very important which, if not addressed will lead to a false sense of security; that is, a situation where the security system is deployed but is either unusable or is deliberately bypassed. A variation of sequential hypothesis testing is proposed to address these issues.

Data plays a very important role in the validation of any new approaches or models that are proposed. Unfortunately, in the user-level intrusion detection domain, due to concerns of privacy, there are only three datasets which are currently available to the research community. This issue is addressed by devising a data generation algorithm called RACOON based on a model used to profile users. The overall goal of this project is to address many outstanding issues concerning user-level intrusion detection and user-level threats, and demonstrate the practicality of proposed solutions. In this light, we have also addressed two related problems, namely, eliciting user cooperation with DRUID through a gracefully degradable QoS model and creating user behavior profiles in GUI based systems for masquerade detection in the Windows environment. The subcontractor Colorado State University has developed a proactive detection and recovery scheme based on attack trees to address certain limitations of the DRUID system.

2 Introduction

Intrusion detection systems (IDS) are becoming increasingly popular because they provide an effective line of defense against attackers who have successfully evaded perimeter defenses such as firewalls. Considering the astronomical growth of network technologies, there are several entry points into an organization and securing each one is not humanly possible, hence, one can expect an occasional unguarded entry point which an attacker has used to gain access. Indeed, if not for intrusion detection systems, the protection provided to an organization's computational infrastructure is very limited.

Intrusion detection systems operate on one or more observables characteristic of a particular attack class and these may appear at several points inside the computational infrastructure. An IDS can be thought of as having two distinct components - 1) a sensor, which performs the task of collecting data, and 2) the decision-making component. If the sensor streams data to be immediately consumed by the decision-making component, then detection is being performed in an online manner. This is a desirable feature since it provides proactive protection. On the other hand, the sensor may output the observed data into a log file to be processed by the decision-making component at a later time. This type of detection is offline in nature and although the protection offered is not instantaneous, it is still called intrusion detection.

One of the first significant advances that has been made at detecting external network level attacks is the development of *Snort* and *Bro*, which monitor network traffic. Recently, advances have been made in identifying other venues of attacks and performing intrusion detection at these locations. Figure 1 gives a high-level overview.

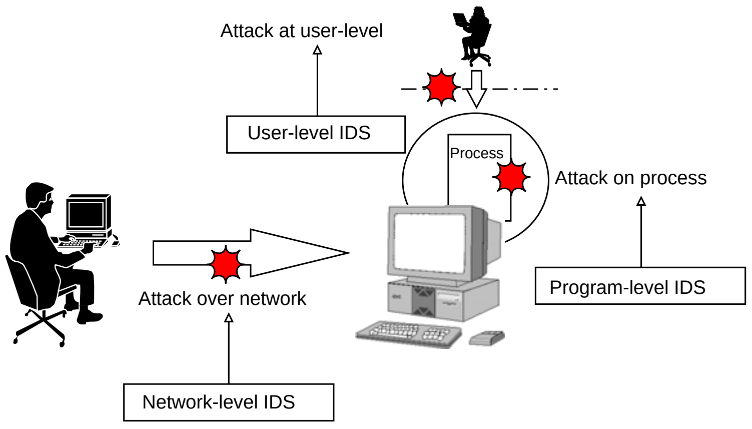


Figure 1: Roles of different types of intrusion detection systems

Network-level IDS. Attackers can target the networking interface of a computer system, either by attacking the networking implementation itself or services which rely on the network to communicate. A network-level IDS looks for hints of these types of attacks and raises alarms when such a pattern is found.

Program-level IDS. The target of an attack can also be a network service through a buffer overflow vulnerability which causes an unexpected jump in the execution flow. When system call traces of normal execution are observed, such an arbitrary jump would appear as a violation. This is the working principle behind a program-level IDS.

User-level IDS. Once an attacker has gained control over a host either through a remote exploit or by a stolen password, he begins unauthorized usage through execution of commands of his choice. In several cases, it is possible to detect a user-level intrusion by monitoring the command trail.

The type of intrusion detection being performed determines wher e the IDS is located. For example, a network IDS is typically deployed at an organization gateway or a host with a network interface configured to operate in the promiscuous mode. Similarly, a program-level IDS is implemented as a host-based system which performs system call interception.

Figure 2 shows the void that this project is trying to fill. Of the three general categories of intrusion detection systems, network-level IDSes are most mature with several commercial grade implementations. For the user-level IDSes to reach that stage, significant advances have to be made. This research identifies several outstanding issues towards the implementation of an online host-based user-level anomaly detection system, which go beyond just measuring performance characteristics. A practical online user-level intrusion detection system is possible only when viewpoints apart from that of an IDS algorithm developer are taken. From the system administrator point of view, there are deployment issues such as the parameters of the system, and the time and effort required to tune them. Similarly, from the point of view of the user who is monitored, it is necessary to keep the annoyances due to false positives to a minimum while providing the necessary detection performance. Only when all these concerns are addressed, can there be a viable user-level intrusion detection system.

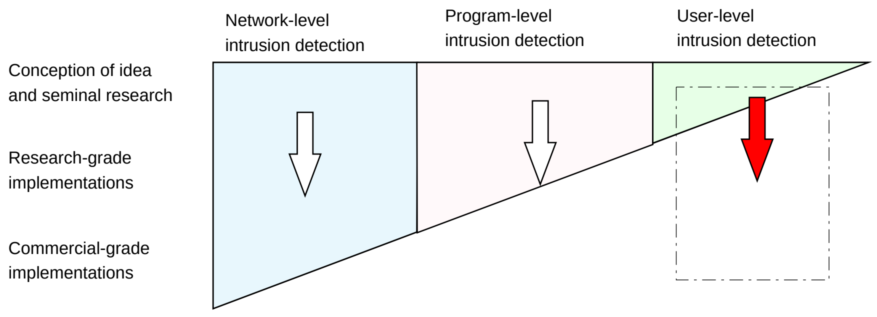

Network-level intrusion detection
Program-level intrusion detection
User-level intrusion detection
Conception of idea and seminal research
Research-grade implementations
Commercial-grade implementations

Figure 2: Overall goal of this research

Currently known intrusion detection systems, apart from their disparate advantages and disadvantages, suffer from a high rate of false positives/negatives and typically cause large space and processing overheads making them less scalable. An ideal intrusion detection system has the following characteristics:

- Stronger deterrence to cracker's attacks by active/online monitoring

- Low latency of detection by rapid decision-making
- Low false positive/negative rate
- Scalable to large and heterogeneous environments.

A careful scrutiny of the above goals reveals a close inter-dependence between each other. An intrusion detection system that can process less information is able to respond faster, make fewer mistakes and scale well due to the lower overheads it generates. However, these are formidable goals and while many claims have been made to achieve them, few have been successful. Therefore, a more aggressive and radical approach is necessary in order to realize these goals. The following stages sum up our effort in this direction:

- A proactive methodology to obtain focused information
- Developing a statistical framework for profiling and a cognitive reasoning support for rapid decision-making with low false positives
- Addressing scalability issues
- Addressing the human-centered security issue and illustrating it in the current IDS
- Extending the concepts to GUI environment that is more commonly in use
- Implementation, testing and performance assessment.

The outcome of this investigation is a user-level IDS which is a comprehensive security management system based on user intent encapsulation through an active query. The owner intent so encapsulated provides the basis for constructing a more focused referenced line to monitor user's activity. Costs based on resource usage and deviation from the known profiles, form the basis for quantification and reasoning about intrusions. Some aspects of user level monitoring do not allow good scalability of the security system and needs attention. Multiple hosts each running the security system can communicate among themselves in order to make a network level decision.

The following section discusses the design and implementation of each of the components of our approach in more detail.

3 Methods, Assumptions and Procedures

3.1 User-Level Anomaly Detection – The DRUID System Development

3.1.1 Intrusion Model and Assumptions

Any activity at the user level is initiated by programs executed on some user's behalf. We classify malicious user activity into the following categories.

- **System abuse and access violations**: A user after logging into a system may execute commands that lower the overall quality of service or attempt to access resources that he is not authorized to.
- **Identity theft attacks**: An attacker can assume the identity of a legitimate user through a password compromise or physically joining an open session left logged on by the user.

In our definition of a distributed system, we include a network of computers that service users on the basis of an account and a password. Further, we assume that the users on different machines have the same user-id although the passwords could be distinct. This model precludes the monitoring of web surfing and anonymous ftp activities. No specific topology is assumed for the network. All communications between nodes are by message passing and the network is assumed to be stable. This model makes our intrusion detection approach unique in that all intrusions are abstracted as happening through well-defined user sessions which are invoked through a user-id and password submission. The problem of intrusion detection simply transforms into monitoring these well-defined user sessions. We also assume that a user session on a node is of finite length. Long periods of inactivity are considered as the end of a session. It is a good security measure to lock the workstation and require reauthentication before unlocking it again.

3.1.2 Basic Principle of User-Level Anomaly Detection

Our technique of intrusion detection based on owner intent encapsulation and verifiable assertions is firmly based on the principle of control flow checking in fault tolerance [1]. In control flow checking, an analysis prior to compilation of the program is done to generate a control flow graph of the application. Signatures or assertions are embedded into the instruction stream at compile time to generate a reference graph. At runtime, the execution is monitored and at designated intervals, the runtime signatures are compared with predetermined signatures of the reference graph. Any discrepancy between the actual signature and the expected signature indicates an error. Both instruction level bit errors and control flow errors are detected by this scheme. Though the control flow checking concept can be extended to intrusion detection, instruction-level models are not applicable here because instruction-level control flow variations may not indicate attacks occurring at higher levels. Accordingly, we use a different approach for the derivation of a reference graph as described below.

In our intrusion detection scheme, the user starts a session on a computer in a standard way, that is, by logging in. The system then encapsulates his intent as a *session-scope*. This is an approximate summary of his intended system usage. Once the scope-file is submitted, the user is allowed to continue with his session. Meanwhile the system translates the scope-file into a set of verifiable statements. When no ordering of events is considered on the activities of the user, the set is simply a table of verifiable statements. It has no control flow information as such.

The verifiable statements give a mechanism for monitoring the user behavior. These statements are generated automatically by reading the scope-file and interpreting the user inputs properly. An important component of our verifiable statements is the subject field. The subject field is generated from the user-id and other unique identifications such as the IP address of the workstation, tty number of the terminal being used etc. All such information will be coded into the subject field. For instance, a user may wish to open multiple login sessions. As long as such intent is expressed in the scope-file, a more general subject coding can be done for this user in order to allow him to work from different terminals or set up multiple login sessions. There is only one monitor process per user even though multiple sessions are opened.

When the user is in session, his operational commands are checked to see if they are the ones he originally intended to execute. Any significant deviation from the plan is an indication of potential intrusive activity.

The flow diagram of Figure 3 represents the basic principle of our new approach and by itself has limited usage. While extensions are easily conceivable for improved performance [2], we retain our basic framework here for ease of presentation of the scheme. Some of the techniques used to minimize false alarms and to build robustness to this basic monitoring scheme are discussed in [2].

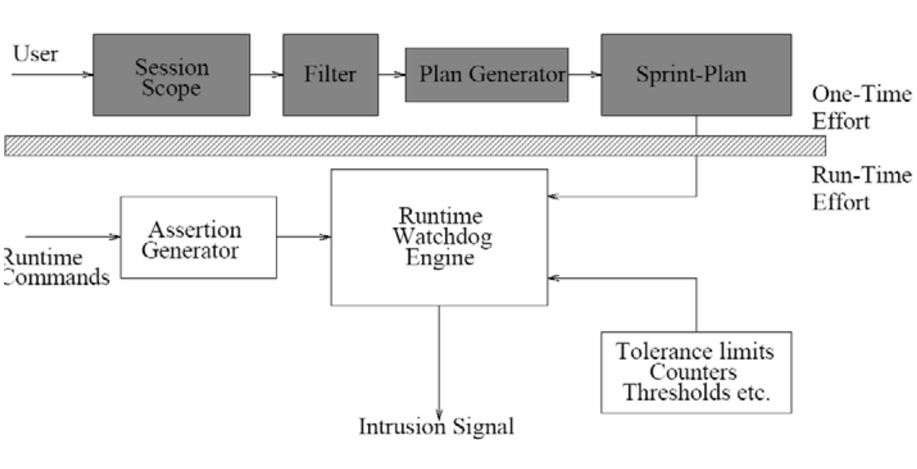


Figure 3: Flow diagram of concurrent intrusion detection system

3.1.3 User Intent Encapsulation

Actively querying a user for computational intent may initially appear as a departure from traditional techniques and an avoidable annoyance, but there are some definite benefits. When an online IDS is installed on a computer system, it makes decisions regarding the current user activity, which may or may not be contested by a user. Figure 4 illustrates the various scenarios.

IDS decision:
Is the current user activity intrusive?

User consent:
Does the user agree?

	Yes	No
Yes	True Positive	True Negative
No	False Positive	False Negative

Figure 4: Scenarios corresponding to an IDS' decisions and a monitored user's response

The four regions, shaded and non-shaded, represent the “hits”' and “misses”' of an IDS' detection mechanism. Perfect intrusion detection is achieved if all the decisions of the IDS lie in the shaded regions without exception. However, that is seldom the case. Misuse detection techniques are generally accurate in detecting known attacks, but they are not complete. On the other hand, although anomaly detection approaches claim to be complete, they are not very accurate on the account of statistical methods being used. Even when the IDS is very accurate, a decision can be wrong simply because the user being monitored contests it. This problem cannot be solved merely by technology or math alone. Instead, we propose to bring the user into the loop. When a user, whether a legitimate user or an intruder, is queried for intent at the beginning of a session, this expressed intent becomes a certificate of normal user activity. Some obvious concerns may arise at this point. This technique can only be practical if the process of intent encapsulation is not very intrusive by nature. Also, it becomes important to do it in a way that captures maximum information with minimum effort.

Let us assume that the entire superset of operations is O as shown in Figure 5. This is the entire set of operations supported on a computer system that any user can attempt to execute.

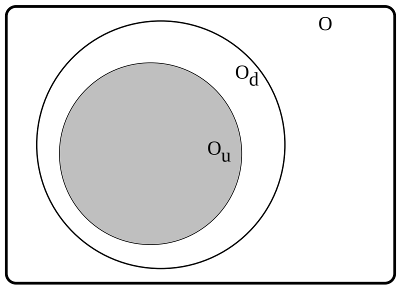


Figure 5: Illustration of effects of implicit and explicit intent encapsulation

3.1.4 Implicit Intent Encapsulation

Observing that certain intents can be inferred directly from the context, we define a default bracket of privileges corresponding to a user's user-id and his role (based on the RBAC methodology [3]). For example, a teller in a bank may have a different job description than that of a manager, therefore requiring a different set of privileges. This bracket O_d can be considered as the hard coded intent of the user. This technique has been widely used in commercial databases and has been successful. RBAC works well when the transactions and the ways in which they are accomplished are few and clearly defined.

3.1.5 Explicit Intent Encapsulation

RBAC based technique suffers from the limitation that these bracket of privileges are static and pre-defined. This results in a very general profile for the users and leads to poor performance.

By explicitly querying the user for intent, it is possible to define a smaller and personalized bracket of privileges or jobs for each user. Let the set of operations that a user defines to be his session-scope be O_u. If this set is bounded only by O, then this technique is not very different from the known statistical techniques for detecting insiders and masqueraders. However, by bounding O_u by O_d instead of O, the user can express intent up to the set defined by his user-id. Such a controlled intent extraction occludes questions such as “What if the user lies?” because the user is allowed to choose only from what he is given. This subset of jobs that are chosen forms the baseline for monitoring through the various sessions. Since this reference line is very focused and small, it becomes feasible to perform online and real-time monitoring which results in a low latency of detection and also lower false positives.

Variations of this technique included querying the user at the beginning of every session. This has the advantage that it can accommodate drastic changes from one session to the other. Its shortcoming is that it is very intrusive by nature. A slightly passive technique involves querying the user only at the beginning of his first session and then invalidating this intent only when some event occurs that warrants another query. For example, a student's activity involving the use of tools for his courses remain unchanged till the end of the current semester. This is less intrusive by nature and is perhaps adequate for most environmental settings.

3.1.6 Summary
The major advantage of intent encapsulation is that it can be done very specific to the environment making anomaly detection very effective. Also, by making the profiles very personalized, it becomes easier to establish the user's identity by way of differences in the choice of jobs and the choice of operations henceforth. More details are found in [4].

3.2 Scalable Implementation of DRUID

Our research prototype DRUID along with an overview of the implementation is shown in Figure 6. This implementation is on the Linux platform with the bulk of the code written using the C programming language. There is a significant emphasis on the user interface and we have both the XFree86 and gtk 2.0 opensource graphics libraries.

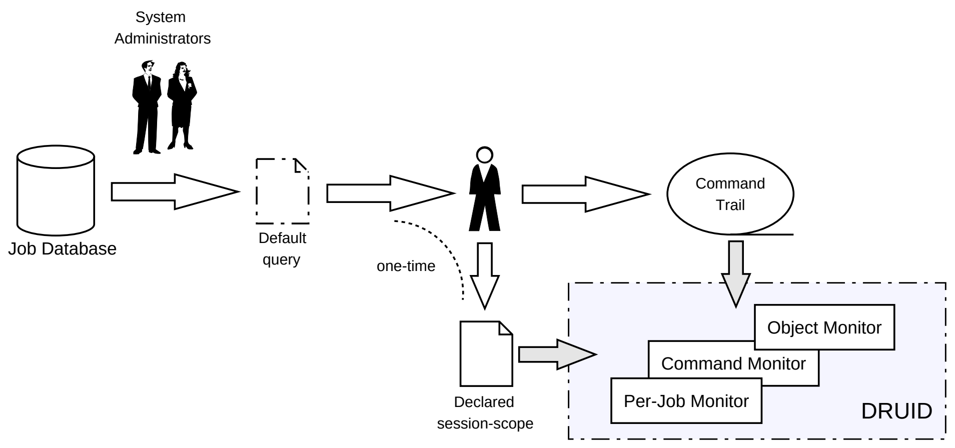


Figure 6: An implementation overview of our approach

The entire process can be divided into two stages - 1) a setup phase, and 2) the actual IDS monitoring phase. In the setup phase, a system administrator determines the relevant set of jobs for a particular user. When the user opens a login session for the very first time, a query containing this default set of jobs, and the constituent meta-commands and commands is displayed. The user is urged to carefully specify the scope of his session. This query is reset and repeated only when there is a reason to believe that the session scope can change drastically. For example, if the role of a user changes, then it might be reasonable to reset the earlier specified session scope and display a new query. The current implementation of this query uses a tabbed window with several checkboxes under every job corresponding to the meta-commands and commands. A job is deemed chosen only when at least one checkbox is selected.

Once the user has completed the session scope specification, he is presented with his user session which has the same appearance as a normal user session would, except that now there are hooks to monitor the user's command usage. There are three main modules within the DRUID IDS, which are:

> **Command Monitor.** This is the main monitoring component which tracks each command executed by the user and the accompanying objects. The current implementation uses ptrace(2) and through system call interception, can monitor and control the execution of every process. It is responsible for ensuring conformance with the specified session scope. Commands which are not specified by the session scope are not allowed and terminated immediately. A process may spawn other processes and these child processes are tracked by following the fork(2) and exec family of system calls.
>
> **Object Monitor.** Objects used in the context of command execution are important for several reasons. Although the jobs, meta-commands and commands are explicitly stated in the session scope, during monitoring this relationship is not very obvious, because all that is seen is a stream of user commands. When a particular command cannot uniquely identify with a job, we defer assigning the command to a job until another command is seen which can unambiguously identify with a job and shares an object with this command. Another important task performed by the object monitor is that it updates the session scope with new commands or executables which are created by the user, for example, as a result of the software development. The object monitor notifies the command monitor that such commands can be allowed except that they are not allowed to maliciously invoke any system calls.
>
> **Per-Job Monitor.** The per-job monitor is responsible for statistical profiling for each job. The command monitor reports commands which have been allowed to execute vis-a-vis the session scope to the per-job monitor with the appropriate job context. However, in cases where it is not able to do so, this task is delegated to the object monitor which waits for several commands before ascertaining the appropriate command-job relationship and reports it to the per-job monitor. Each job is pre-specified with the relevant model and the per-job monitor builds statistics based on this model. When a preset threshold of data is reached, then the detection capability is switched on and any further incoming data is compared with constructed profile.

3.2.1 Summary

The overall detection methodology requires the cooperation of the user. When the user is certain that the current session is the very first one but there has been no query, then it is a cause for serious concern since someone else has already accessed the account and responded to the session scope query. If the very first session is attended to by a legitimate user, any session accesses from the adversary are likely to cause the IDS to raise an alarm very rapidly since the adversary is not aware of what the legitimate user has specified in his session scope. Also, owing to the session scope query, there is very little room for ambiguity, which an attacker can take advantage of.

3.3 Addressing Deployment Time Issues

Addressing the deployment time issues of an IDS requires a comprehensive solution which goes beyond just performance enhancements. Towards this overall goal, a technique known in the statistics domain called *sequential hypothesis testing* (SHT) [5], [6] is leveraged. This technique was originally proposed to take the sequential arrival of data into account, making decisions as the data arrives, with the possibility of determining very early whether the currently observed data conforms to the expected data model, while maintaining the same performance specifications. SHT has found successful applications in other areas of computer systems security such as worm detection [7]. Although the basic form of SHT can address our first two goals, *viz.*, online detection and usability, it is still susceptible to mimicry attacks, and we have addressed this problem through modifications to SHT.

3.3.1 Using Sequential Hypothesis Testing

As mentioned earlier, SHT can solve two very important problems towards an online host-based intrusion detection system. First, since decision-making is performed sequentially as the data arrives, it makes for a natural online algorithm. Secondly, SHT guarantees the user-specified bounds α and β on false positive and detection rates. We now briefly discuss the 1-step Markov Model [8] and show how it fits in the SHT framework.

1-step Markov Model [8]: In the 1-step Markov Model, statistical inferences are performed using a Bayesian analysis, which is a fairly involved process, and hence, we include only the required details here. There are two competing models, one which assumes a Markov Model and that observed transition probabilities are consistent with the historical transition matrix, and the second which assumes that they are generated from a Dirichlet distribution. The null and alternative hypothesis are formulated as follows:

$$\text{null hypothesis } H_0 : P(X_t = k | X_{t-1} = j) = p_{jk}$$

$$\text{alternative hypothesis } H_1 : P(X_t = k | X_{t-1} = j) = Q_k$$

$$\text{s.t. } (Q_1, Q_2, \ldots, Q_k) \sim Dirichlet(\alpha_1, \alpha_2, \ldots, \alpha_k)$$

where $p_{jk} = P(\text{next command} = k | \text{ previous command} = j)$, and all α_i are estimated by fitting marginal frequencies to a Dirichlet distribution as explained in [8].

The Bayes Factor (BF) is calculated over a block of data X as:

$$BF = \frac{P(X|H_1)}{P(X|H_0)}$$

which is simply the likelihood ratio Λ. When BF is very large, then there is greater evidence supporting the rejection of the null hypothesis, or in other words, the current user activity does not correspond to the legitimate user's historical profile, and an alert can be raised.

1-step Markov Model (SHT version): When implementing the above technique using SHT, the two hypotheses are kept unchanged. However, an additional region of uncertainty $H_?$ is added.

$$\text{null hypothesis } H_0 : P(X_t = k|X_{t-1} = j) = p_{jk}$$

$$\text{region of uncertainty } H_? : \text{need more data}$$

$$\text{alternative hypothesis } H_1 : P(X_t = k|X_{t-1} = j) = Q_k$$

$$\text{s.t. } (Q_1,Q_2, \ldots ,Q_k) \sim Dirichlet(\alpha_1,\alpha_2, \ldots ,\alpha_k)$$

Let α and β be the user-specified bounds of false positive and detection rates. Assume that events $X_1, X_2, \ldots , X_t$ have arrived so far. Then, using the likelihood ratios, statistical inference proceeds as follows.

$$\frac{1-\beta}{1-\alpha} < \Lambda \Rightarrow H_0$$

$$\frac{1-\beta}{1-\alpha} \leq \Lambda \leq \frac{\beta}{\alpha} \Rightarrow H_?$$

$$\Lambda > \frac{\beta}{\alpha} \Rightarrow H_1$$

In the usual SHT setting, the experiment stops as soon as the likelihood ratio moves into one of either H_0 or H_1. However, for an intrusion (masquerade) detection system, the experiment has to be restarted if the likelihood ratio converges to H_0, because more data may become available as a result of the user executing commands.

3.3.2 Randomized Sequential Hypothesis Testing

Since SHT performs statistical inference with each arriving data event, attacks against data aggregation are unlikely to succeed. However, SHT is still a statistical methodology and is vulnerable to other types of mimicry attacks; in particular, an attacker can indefinitely hide within the region of uncertainty. How an attacker can launch such an attack and elude detection is described in detail in [9].

Revisions to SHT have also been proposed in [9] by including randomizations such that not only do the operational characteristics of the masquerade detection system appear random, but also the region of uncertainty collapses, leaving no room for attacker to hide.

3.4 Data Generation

User level intrusion detection deals with user command data, which is closely tied to user behaviors. The process of collecting or generating data suffers from similar issues as network level data if not worse. Obtaining a representative data set for each user takes months or some times years. Lane [10] reported that their data collection process took nearly two years for only eight users. Not all of this data could be used because of errors and requirement of anonymization. As a result, the data set had to be sanitized. In general, the data collection process involves setting up a command monitoring tool on the user's computer and this could be looked upon as being intrusive to the user. Also, it doesn't guarantee good data because the user's behavior may alter if he is aware of the fact that his commands are being monitored. Though there are simulation tools available in the networking domain such as OPNET and ns2 and data generation tools such as LARIAT of MIT Lincoln Laboratory to aid researchers, and none are available for user command data. In view of these issues, a user command data generation tool called RACOON is proposed to expedite the process of development and evaluation of user-level intrusion detection. The two main aspects of our approach are:

Job-centric Approach. RACOON works on the assumption that a user's computational behavior is a causal process indicated by the commands he uses to accomplish a job or task. Notable works such as that by Lane and Brodley [11] have used this assumption as the basis of most of their work; it is just restated here. There could be different classes of users such as programmers, scientists and system administrators, each with different job preferences and peculiarities in user command usage. In our approach, the notion of jobs is used as a second order description of user behavior; the first order being the commands itself. First, this model to capture user behavior is described and then shown that data generated using this model is very similar to an actual user command data set.

Customizable Templates. One of the important aspects of any data generation tool is the support for "tunability" of data. Data of varying quality not otherwise seen in the wild can be generated, and used to evaluate and expose the blind spots of an IDS. A template in RACOON represents a particular user profile. It contains the parameters used to replicate user behavior. Since these parameters are user-controlled, it is possible to customize each template to reflect a particular user profile. Subsequently, one can generate data which is pertinent to a particular computational environment and with required noise levels.

Figure 7 shows the overview of RACOON. There are two paths for generating user command data. In the first path, a user specified template is created and provided as input to the data generation module. Manual specification can be a onerous process and a front-end to assist the user is being implemented. In the second path, an available data set can be processed to create the template, which then follows the first path. Several user-controlled parameters such as data size allow the generation of data of desired size and quality. In order to evaluate our tool, data sets resembling Schonlau's data set [12] consisting of 750,000 commands were generated in a matter of a few seconds, and statistical similarity tests were performed between the two data sets - generated data and Schonlau's data. Finally, some well-known anomaly detection algorithms were duplicated and the two data sets benchmarked against them.

We will provide the results of evaluation of RACOON in Section 4. Technical details of RACOON are found in [13].

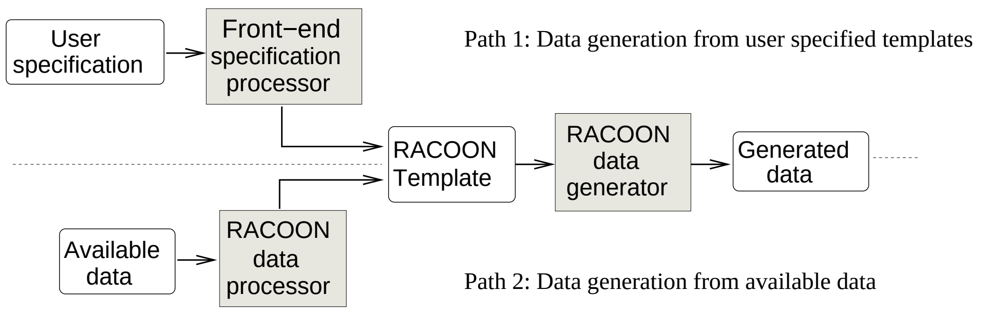

User specification
Front−end specification processor
Path 1: Data generation from user specified templates
RACOON Template
RACOON data generator
Generated data
Available data
RACOON data processor
Path 2: Data generation from available data

Figure 7: Overview of RACOON's data generation process

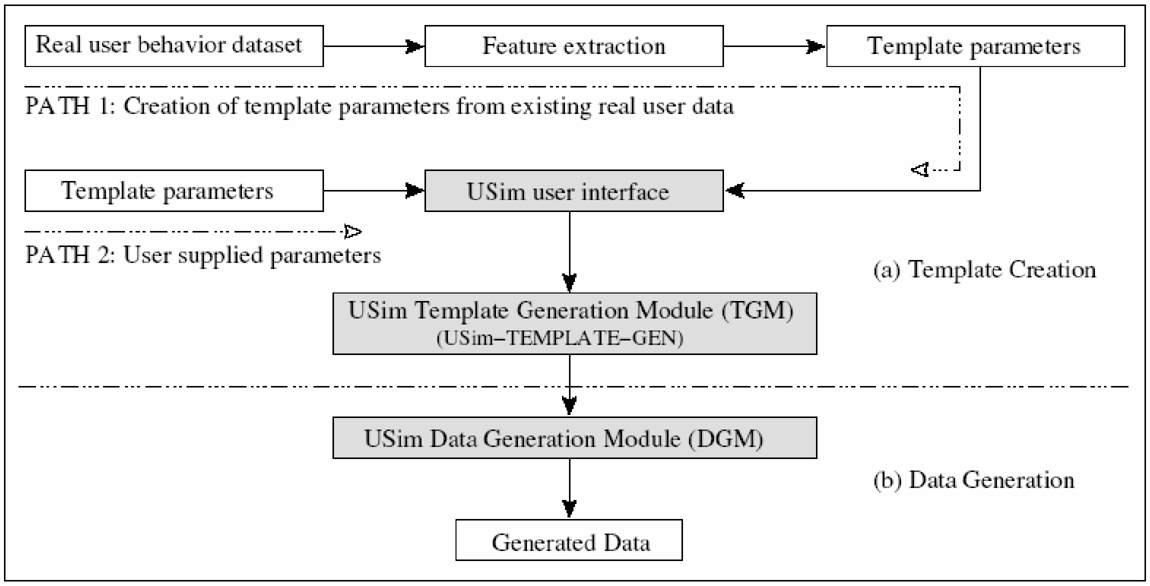
3.5 DRUID Extension to GUI Based Systems

Due to the use of GUI-based platforms in target environments like military and industrial organizations, a need has arisen to port DRUID to the Windows Platform. This requirement has brought about new challenges in generating profiles for users in the graphical environment. Therefore, we have developed a new framework called USim for data generation, training and testing of DRUID in a GUI environment. In this framework, we automate the generation of user data by parameterizing user behavior in terms of user intention (malicious/normal), user skill level, set of applications installed on a machine, mouse movement and keyboard activity. The user behavior parameters are used to generate templates, which can be further customized. Similar to RACOON, the framework achieves rapid generation of user behavior data based on these templates on GUI based systems. The data thus generated can be utilized for rapidly training and testing intrusion detection systems (IDSes) and improving their detection precision. This framework can also benefit research where user behavior data is utilized to improve usability and quality of software products. One such application is in the Human-Computer Interaction (HCI) domain, where the proposed technique can provide better testing capabilities.

3.5.1 USim Design

USim is a specialized tool designed for generating user behavior data. USim generates this data based on provided customizable templates. There are two ways a template can be created in USim. First, the proposed framework can take an existing real user behavior data and extract parameters from this data for creating templates. Secondly, USim can also be fed precise values of the parameters of a simulated user for creating templates. Figure 8 shows these two paths of the USim framework and data generation methodology. This methodology can be used to rapidly generate large volumes of user behavior data. This data is needed for thorough training and testing of intrusion detection systems.

When USim is supplied with the real user behavior dataset (see PATH 1 in Figure 8), it extracts the simulation parameters from this dataset. These parameters are passed to the USim template generation module (TGM). These parameters could consist of, but are not limited to, user behavior (normal/malicious), user session scope (type of applications being used), user role (admin/developer) etc. Each parameter is specified with a probability and modeled as appropriate for the required user behavior. TGM module generates USim understandable template and sends them to the data generation module of USim (DGM). USim uses XML for representing templates due to its wide popularity and cross-platform compatibility. Once DGM receives the templates, it generates appropriate data based on the provided parameters. The second branch of USim (see PATH 2 in Figure 8), deals with the parameters directly passed for creating templates. These parameters are expected to be as precise as possible and represent a type of users (like a group of developers or graphics designer). The created template can be tuned later on, depending on the type of user being simulated.



Real user behavior dataset
Feature extraction
Template parameters
PATH 1: Creation of template parameters from existing real user data
Template parameters
USim user interface
PATH 2: User supplied parameters
(a) Template Creation
USim Template Generation Module (TGM)
(USim–TEMPLATE–GEN)
USim Data Generation Module (DGM)
(b) Data Generation
Generated Data

Figure 8: Data generation using USim

3.5.2 Real User Behavior Data

Data collected during a user's activities on a system in real-time is called *real user behavior data*. This dataset describes a user's behavior according to his/her job assignment. The real user behavior data is utilized by the USim framework for extracting features and creating templates for generating similar data. The generated data in this case should closely resemble the real data.

3.5.3 Dataset Structure

Structure of the data generated using USim framework is shown in Figure 9. As can be seen from the figure, the dataset contains details about user behavior such as executed commands, mouse and keyboard activities, currently running background processes etc. The mannerism of user behavior provides information about deviation from normal behavior and is a useful feature for improving detection accuracy.

3.5.4 Simulated User Behavior

Another approach taken by USim is generating user behavior data based on specifications provided to the framework. The specifications are converted to templates, which are customizable and can be tuned for a specific purpose. These templates contain information such as user's job function, type of user, mannerisms of keystrokes and mouse movements etc. The data thus generated is said to be for a *simulated user*.

Modeling user behavior anomaly detection mechanisms rely heavily on the understanding of user's behavior for detecting malicious activity effectively. User behavior is a complex process and to be able to efficiently model it, we identified some key features from the Human factors domain such as nervousness while using the computer, keyboard and mouse activity, typing speed etc. [14]. These features help explain USim framework and put our work in perspective.

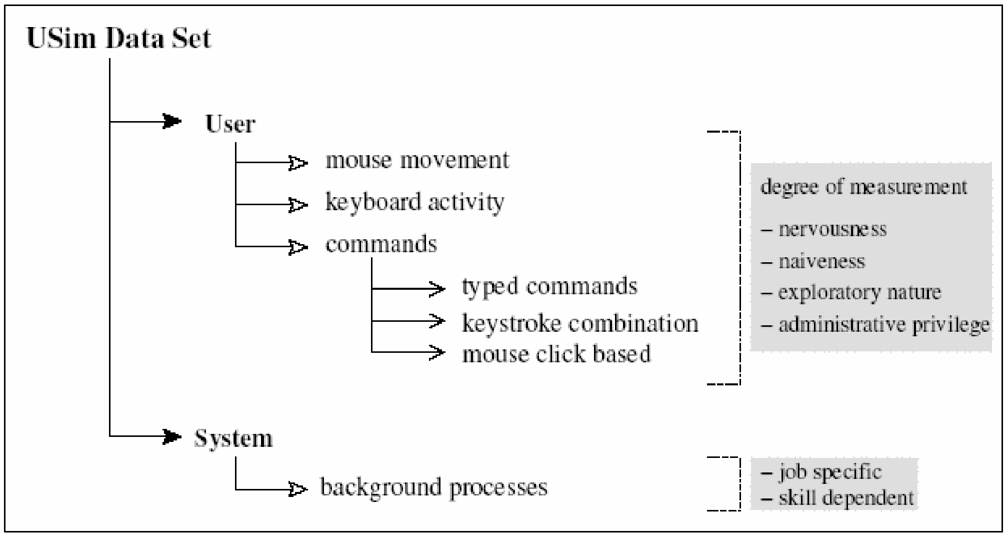


Figure 9: USim dataset structure

Command: The most basic entity to achieve a certain goal on a computer is called *command*. A command is the most prominent feature in the user behavior dataset. It describes the goal and hence user intention. Example of a command is ls and vi on Unix.

Meta-command: The category of a command is called *Meta-command*. Commands related to the same goal are collectively categorized as one meta command. For example vi and emacs belong to editor meta-command.

Session-scope: The set of meta-commands to achieve a set of goals pertaining to a user’s role is called his/her session-scope. For example, a sample session-scope of a user, whose role is *developer*, would be meta-commands: editor, compiler, linker, debugger. It should be noted that the above three features are referred from our previous work in [13].

User role: A user's role represents the scope of his/her responsibility. User's role also dictates his/her session-scope.

Key-time: Usual time difference between keystrokes for a particular user's *normal* profile is called *Key-time*. It should be noted that this is a very useful feature and in limited logging scenarios may be used for separating masqueraders from normal users.

Mouse-graph: The movement of mouse based on a user's level of nervousness. A high mouse-graph may either mean high nervousness of an insider or high anxiety level for a masquerader.

Posture and body language: The way a user is sitting while working also plays an important role in determining his level of comfort while working. Again a masquerader won't be too much concerned about his posture or body language but a normal user would first try to seat himself properly and then start working. This particular feature although very useful in determining a malicious user physically, is out of scope of this type of framework and hence will not be considered in our model. It is provided here for the sake of completeness of the model.

3.5.5 User Behavior

User behavior in cyber domain can be described as the way a user performs his activities on a computer system. This includes the applications and resources utilized, the temporal relationships between the various activities performed on the system (time between various commands, keystrokes etc.), the movement of mouse (a nervous/malicious user might move the mouse more rapidly than a normal user). USim aims to simulate this user behavior (also called the user behavior *profile*) and provide realistic datasets representing this behavior. The advantages of such datasets are many-fold. They can help in determining the efficacy of the systems algorithms, the response of the system to such users and the performance of the system. User behavior can be defined with the help of two classes of parameters. First, the system settings such as mouse movement speed setup, left handed or right handed mouse, individual application being used and screen resolution. Secondly, the environmental/technical factors like knowledge of the peripheral device being used, user's comfort level with the system and expertise with the particular application being used. A good example of behavioral differences in two people is a secretary in an organization versus a graphics designer. While the secretary would be utilizing the keyboard more for typing the documents and memos, the graphics designer would be involved in heavy mouse movement activities due to the nature of his work. The overall approach of USim framework is to provide customizable templates to capture this user behavior and help in generating datasets to represent the behavior.

3.5.6 Template Generation

We consider a particular template that satisfies the requirements of most of the anomaly detection algorithms available today. Consider a desktop system with a user working on it to fulfill a given job requirement. We consider a Microsoft Windows environment for illustration purposes, although the template can easily be modified to suit any operating system. Most of the anomaly detection schemes require a series of commands that have been executed by the user. At least, till date, the data that has been fed into such algorithms have been that way, i.e., a list of commands. Such lists, while adequately representative of a UNIX system of the past decade, are no longer meaningfully representative of the systems today. Graphical user interfaces (GUIs) have become the norm for most of the computer systems today. Hence user characteristics are better represented by datasets that are more expansive and take into account the user's operation on a GUI system rather than just the command list. Simulating such behavior has unique challenges and is an issue that has never been addressed before. A typical Microsoft system has user-run processes and system-run background processes. The load on the system, which includes both these processes, also varies according to the users’ command execution. While we say command execution, we include the applications run by the user such as MS Word and Internet Explorer in addition to the usual command prompt tools/utilities. Furthermore, two additional user characteristics are included, which are the mouse movement and the keyboard typing speed. These characteristics have largely been ignored by the simulation tools so far, which is surprising since they are the closest we can get to the user characteristics apart from the commands they execute. Below we describe in further detail the template for generating dataset encompassing such characteristics:

- The first manner in which we can generate simulated data is from a set of real data. Real data for various reasons, privacy being the foremost, are hard to come by for a system. However, in cases when they are available, USim can take two inputs viz. the real data and a template that effectively describes the dataset and an expected distribution of the dataset features. With sufficient amount of real data, we do not require the expected distribution of the dataset features. This methodology of generation is similar to our prior work in [13].

- Secondly, we describe a customizable template that can be used to generate user data taking into account the GUI characteristics of the system. As mentioned before, mouse movements and keyboard typing characteristics are the two features that come very close to representing the user (almost in a biometric manner, similar to signatures and thumb-prints). This template can be used for training and testing any intrusion detection system. The template also specifies the roles of the user and can provide precise behavior profiles accordingly. The template for a graphics designer will have more graphics applications as compared to an administrator. Hence USim can generate the mouse movements and command streams accordingly (adhering to profile, exploratory user or malicious user). In addition, the system background processes and the load on the system also change correspondingly. A complicated graphics rendering engine might take a huge amount of load (processing overhead). A system utility may also generate a huge load. However when a system utility is executed, background processes with the system credentials (as opposed to the administrator's credentials) will be fired up. The template specifies these criteria also and USim generates the command stream appropriately. This section thus illustrates how flexible template generation is through the USim architecture.

An algorithm called USim-TEMPLATE-GEN, which can generate templates suitable for USim framework for generating user behavior datasets has been developed. More details can be found in [15]).

3.6 Eliciting User Cooperation for Enhanced Security in DRUID

We have investigated an interesting and very relevant problem in DRUID in relation to its core concept viz., a user stated session-scope. Given that the human factor is the weakest link in most security systems, we found that DRUID was vulnerable since it relied on the user to provide an accurate session-scope, which is essential for effective intrusion detection. To elicit user cooperation, we investigated certain psychological factors of humans (cognitive aspects of reasoning) and proposed a model that varies the application level QoS experienced by the user based on the accuracy of his session-scope. The lowered QoS in the face of an inaccurate session scope (too broad or too narrow), we argue, will encourage users to provide an accurate session scope and thus maintain the security level of the system.

During the last few months of this contract, we extended our ideas in two directions. In addition to reducing application level QoS, we are building a notion of 'interruptive interfaces' where a non-cooperative user is frequently interrupted to encourage him to perform the expected action by the security subsystem. Additionally, we have introduced the notion of learning the distribution of unknown factors to reduce the risk level from the weak human factor. This concept basically attempts to learn about the threat models in a system that are initially unknown, but have the common feature of exploiting the weak human factor. This work is a prelude to two other very fundamental works: (a) building an abstract risk assessment model for the weak human factor and (b) attempting to generate concrete situational awareness through a practical implementation of counter-example guided controls.

This is an exploratory research with significant potential to address the weak human factor in the security equation. This concept will be further developed as future work.

3.7 Proactive Detection and Early Warning

Colorado State University (subcontractor) has addressed a significant concern by observing that most IDSes including DRUID generate alerts only after they are able to see the misuse signatures or some deviations from what is normally expected. A malicious activity may result from a sequence of perfectly innocuous activities. Intrusion detection systems do not report on these activities mostly to prevent information overload for the system administrator. Thus an intrusion detection system generates an alarm only after the cause for alarm has occurred. In many situations however, this may already be too late.

We propose a new approach that can be used to predict attacks arising from an insider’s activities. This work uses the user-intent analysis approach of DRUID which ensures that during a particular session a user remains reasonably within the scope of a previously declared set of activities. Any digression beyond this reasonable limit constitutes a misuse of system and steps are taken to protect against such digressions. However, this approach fails to account for the fact that a user may remain completely within the scope of a previously declared set of activities and still be able to launch attacks. This is where this extension work contributes.

We begin by assuming that it is possible to enumerate the different attacks that a user can launch against a given system. This assumption is not unreasonable for known attacks. Almost all network vulnerability scanners provide such information. We then determine all the possible actions by which a user can launch an attack against the system. We map these actions against a user’s session-scope to identify which sequences of these actions can potentially lead to system compromise. Next we monitor each user’s activities to see if and how they match against these sequences. Depending on the match we propose an estimator of attack probability. Our approach is different from classical intrusion detection systems. It works as an early warning system. We continuously provide the system administrator an estimator of attack probability. Thus we cannot associate a rate of false positives or negatives with our technique. Our objective is to ensure that the system enters an alert mode once the probability of an attack is determined to be sufficiently strong. The notion of “sufficiently strong” is based on perceived risks. In the alert state, the following actions are undertaken to ensure the survivability of information in case of an actual attack.

- Allocate additional resources to assist in data collection by logging system wide activities more aggressively, saving system states more frequently and initiating recovery contingency plans by coordinating with other monitors.
- Re-distribute essential services to other safer portions of the network.
- Introduce mechanisms to handle possible attacks including ways to contain the attack.

All these activities are continued until either an intrusion is actually signaled by accompanying intrusion detection system or no further signs of attack are identified. The advantage of this approach is that it is a flexible and resource efficient technique for security management. At the same time it is a guarded approach. If an attack succeeds (which is determined by techniques other than ours), it allows the system to be in a fully prepared mode for subsequent recovery.

The formal definitions, algorithms and some theoretical results regarding proactive detection and recovery can be found in [16].

3.8 Porting DRUID to Windows

In order to enhance the application value of DRUID, it has been ported to the Windows platform. We have broken the DRUID implementation down to the following components:

Role Specification Module: This module is invoked at the start of the deployment of DRUID. The administrator uses this tool to specify the applications that a user of a given role can choose. The Role specification module takes as input a jobs database that has a comprehensive list of jobs for all possible roles in an organization. It outputs a session scope query (XML) file that lists the choices that will be presented to a user.

Session Scope Specification: The session scope specification module is presented to the user at the start of his session. This is a one time module that queries the user of his intent during the session. It takes as input the session scope query file (output by the role specification module) and presents the user with an appropriate interface.

System Logger: The system logger & QoS monitoring engine constitutes the core of our model implementation on DRUID. It is a Windows service that is run when the machine starts up. It is run with the local system's credentials. The system logger monitors the system and receives a notification every time a Win32 process starts up and shuts down. It acts on this notification and writes relevant details about the process in a log file.

QoS monitoring Engine: This module is based on the concept of eliciting user cooperation for enhanced security. The QoS monitoring engine receives notifications from the system logger and compares the process information history with the stated session scope. The application level QoS rendered by the system is modified according to the result of the comparison.

At the time of completion of this project, we were working on the following features for DRUID:

- A plug-in type architecture so that future developers can add extensions to DRUID if they so wish.
- Tighter integration between the components and a singe installer for testing purposes.

Latest information on the prototype status and corresponding downloads can be found at http://www.cse.buffalo.edu/caeiae/projects/DRUID.

The next section gives details of evaluation of DRUID and its various components.

4 Results and Discussion

We have tested the various components of DRUID through elaborate simulations and the results of our performance study are presented in this section.

4.1 Preliminary Experiments on DRUID

We chose a student/faculty user academic environment for the purposes of testing the DRUID framework. This environment has fewer security controls in place and allows greater freedom for the users and hence makes a good testbed to study the efficacy of our technique. We could have also considered a regulated system such as web-enabled banking to estimate the feasibility and impact on a more general and commercial setting.

4.1.1 Simulating a University Environment

The basic architecture is client-server based. Such a setup allows us to derive some test cases from the published descriptions of well known attacks and in developing site-specific test cases based on the security policy. It also helps us to consider both sequential and concurrent intrusions. In a sequential intrusion, a single person issues a single sequence of commands from a single terminal or a workstation window. In concurrent intrusion, one or more intruders issue sequences of commands from several terminals, computers or windows. The command sequences work cooperatively to carry out an attack. For example an intruder can open multiple windows on a workstation and connect to a target computer from each window and try to distribute his intrusive activities among them. The platform allows us to simulate basic sessions such as telnet, ftp etc. Synchronization can be achieved which lets us specify a fixed execution order of events.

When the student/faculty user logs in with a user-id/password submission, password verification is done first. If the user is authenticated to login he will be provided with a series of GUI windows to specify the scope of the session. The user selects the application he is going to work on. If, say, the user selects Research as the application, the user is provided with a pre-selected input list containing various categories such as simulators, design tools, operating systems, programming languages, scripts, documentation and miscellaneous items such as ftp, rlogin etc.

The user just needs to check the tasks he intends to perform. Once this is done the watchdog queries the user if he intends to perform any other activities that are not present in the predetermined list. The user is also queried if he intends to open multiple sessions.

The various components of the session scope are combined together by a formatter to obtain the intent in the final form. Figure 10 shows a run-time monitoring setup for a 1-user, 2-hosts system on a single server and is explained below.

The plan (called sprint-plan) generated for the user is stored at a secure location on the server. As soon as the user logs in to a host, the watchdog checks to see if there is a sprint-plan already existing for the user, if there is none, it generates a new one. If a sprint-plan already exists, the user is allowed to proceed with his normal activity. The watchdog continuously monitors the user and compares it with the sprint-plan.

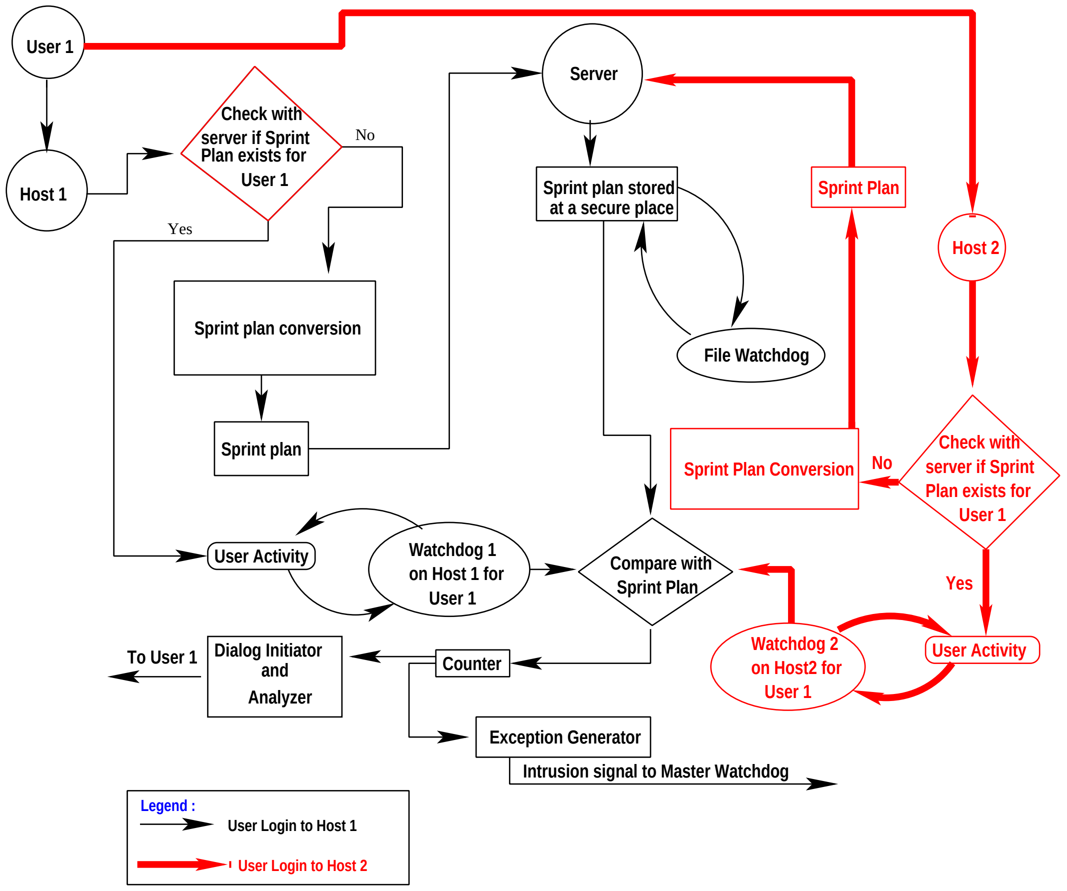

User 1
Host 1
Check with server if Sprint Plan exists for User 1
No
Yes
Server
Sprint plan stored at a secure place
Sprint Plan
Host 2
Sprint plan conversion
File Watchdog
Sprint plan
Sprint Plan Conversion
No
Check with server if Sprint Plan exists for User 1
User Activity
Watchdog 1 on Host 1 for User 1
Compare with Sprint Plan
Yes
To User 1
Dialog Initiator and Analyzer
Counter
Watchdog 2 on Host2 for User 1
User Activity
Exception Generator
Intrusion signal to Master Watchdog
Legend :
User Login to Host 1
User Login to Host 2

Figure 10: Run-time monitoring setup

4.1.2 Test Cases and Attack Scenarios

The test data is based on user activity collected over a period of two months. We required the test data to be confined within the same semester. We used this data to derive the verifiable assertions and then test the strength of the scheme by subjecting it to a few test cases and attacks.

There is usually no simple procedure to identify appropriate test cases for an intrusion detection system. A variety of intrusion scenarios are considered based on some common practices of system usage. These scenarios are grouped into four categories, viz., one-user without multiple logins, one-user with multiple logins, multiple users without multiple logins and multiple users with multiple logins. Two set of experiments are performed in each of these categories, first with the worst case, where a user selects all the entities provided in the session-scope GUI by the watchdog and the second where a user selects only a few entities. The tests are performed by treating the logins as four different cases, with up to two users at a given time. The first case is where both logins are legitimate. In the second case, the first login is from a legitimate user and the second login is from an intruder. In the third case, the first login is from the intruder and the second login is from the user and finally the fourth case where both logins are from intruders.

We performed a total of 32 attacks and they were of two types. One category represented very obvious and apparent attacks such as transferring the /etc/passwd file from one host to another, password-cracking by comparing the entries in the /etc/passwd file to entries in another file, using a dictionary file for the same, and exploiting the vulnerabilities such as rdist, perl 5.0.1, etc. The system is able to detect all the intrusive activities and terminate the connection for the logins of intrusive users. The second category involved more subtle attacks similar to mimicry attacks [17]. Even in such cases, since we monitor both the operations and the file system accesses, we are able to restrict the damage caused by the intruder. The intruder is only able to cause damage within the user's login and home directory. In the worst case scenarios of one-user with multiple logins and multiple users with multiple logins, a relatively larger number of intrusive activities was not detected. The system has also generated a few false positives, flagging an intrusion when normal user activity is taking place. This happens when the user selects only a few entities from the session-scope. The results are summarized in Table 1 where detection latency is reported in terms of average number of user operations. The metrics shown in the table seem to be consistent with intuitive predictions.

Table 1: Summary of preliminary simulation results

Sessions	Metrics	1 User, No Multiple Logins	1 User, Multiple Logins	2 Users, No Multiple Logins	2 Users, Multiple Logins
User And User	Detection	-	78.6%	74.9%	91.9%
	Latency	-	35	36.1	29
	False Positives	-	21.4%	25.1%	8.1%
	False Negatives	-	0%	0%	0%
User And Intruder	Detection	98%	89.0%	100.0%	94.7%
	Latency	0	11	0	9.6%
	False Positives	0%	0%	0%	0%
	False Negatives	2%	11%	0%	5.3%
Intruder And User	Detection	99%	100%	98.2%	100%
	Latency	0.4	0.7	0.6	0.5
	False Positives	0%	0%	0%	0%
	False Negatives	1.4%	0%	1.8%	0%
Intruder And Intruder	Detection	58%	81.3%	77.4%	91.5%
	Latency	15.9	14.8	17	27
	False Positives	0%	0%	0%	0%
	False Negatives	44%	18.7%	22.6%	8.5%

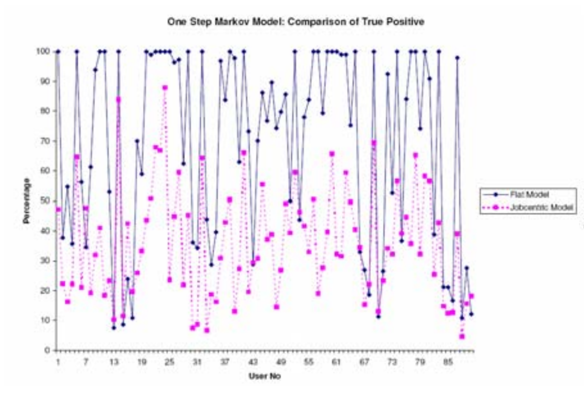
4.1.3 System Overheads and Performance Impact

Since Java is used for implementation, moderate impact on system performance is expected. When new connections are made or more users login, the system load increases. However, this increase is only marginal because there is no need to maintain any large data structures for each user or connection. The main server on which our intrusion detection system is running is a Sun Ultra Enterprise 450 Model 4400 and the clients are Sun Ultra 5's running Solaris 2.7.

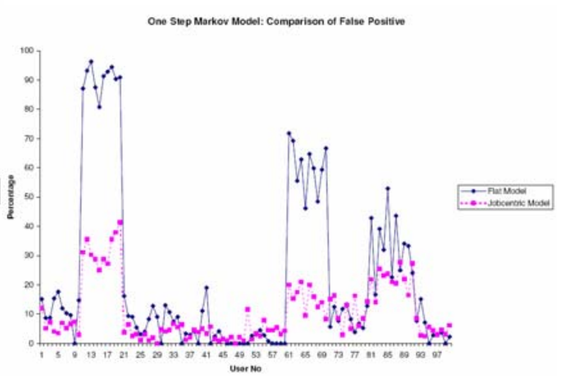
A normal user in a university environment is assumed to have about six to eight processes running on the system at a given time. There is one watchdog dedicated for each user which makes it one more process per user on the system. This watchdog process does not use many run-time resources and hence may not become an overhead to the system. However, when several users are logged in and are being monitored, the system may see some performance loss. In order to study this overhead, we eliminated all unrelated activities in the test environment, started the intrusion detection system and allowed the users to log in. We analyzed the average load per minute (no. of jobs in the run queue on Unix) and the storage overhead in kB against the number of users on the system. In our preliminary implementation without much optimization, the operation is very stable for about 15 users. The load on the system tends to increase as the number of monitored users increases beyond 15. The storage overhead (325 kB for a single user) increases at a constant rate with the number of users. When the session-scope is large, the watchdog maps it to a huge sprint-plan. The storage used by the IDS in our study corresponds to the worst case scenario where a user selects all the entities from the session-scope provided by the watchdog in a GUI.

4.2 Evaluation of Job-Centric Approach

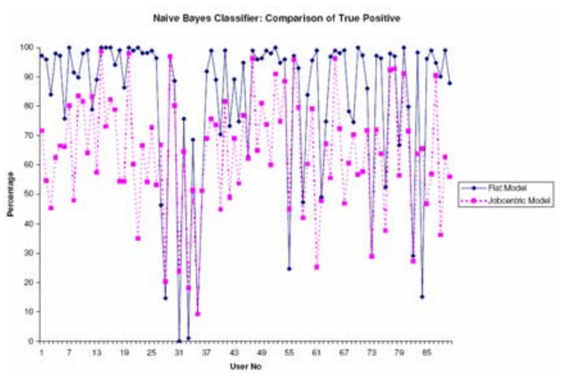
Most of the currently known approaches to anomaly detection have been benchmarked against Schonlau's dataset. The dataset consists of user command trails from 50 users, each 15,000 commands in size. The standard experimental setup involves dividing each the command set of each user into two parts - one of 5,000 commands for the purposes of training, and the other of 10,000 commands. The second part is polluted with blocks from other users. The main idea of the experiment is to determine how many of these blocks are detected by the IDS and provide an estimate of the detection and false positive rates. In the context of our approach, the knowledge of jobs is very important. However, Schonlau's dataset is job-agnostic and has no job-related information whatsoever. Another dataset was not used since the known techniques have been benchmarked against this dataset. This pitfall was overcome by manually inspecting the dataset and inferring potential jobs based on the commands. Also, since the commands were collected on an old UNIX variant, some of the commands do not have modern equivalents, and finding their purpose was difficult; in some cases, inferences had to be drawn from the names of these commands. We have evaluated the job-centric approach using both frequency-based (Naive Bayes Classifier) and sequence-based (One Step Markov Model) approaches. Figure 11 shows the detection and false positive rates of both these approaches. Notably, the false positive rate in both these approaches is significantly lower. Moreover, the detection rates are also comparable, although slightly lower in some cases. The latter can be explained by the fact that dividing the data into jobs, effectively reduced the amount of data per job but increased the number of test cases. Both these factors affected the detection rate. For example, the flat text-based approach may have a detection rate of 60% (2/3), while our approach has only 50% (15/30), but the test cases are far more. This can be construed more as a limitation of the dataset rather than our approach. In fact, we believe that our approach would have outperformed a flat text-based interpretation, had the dataset been in a job-aware format.



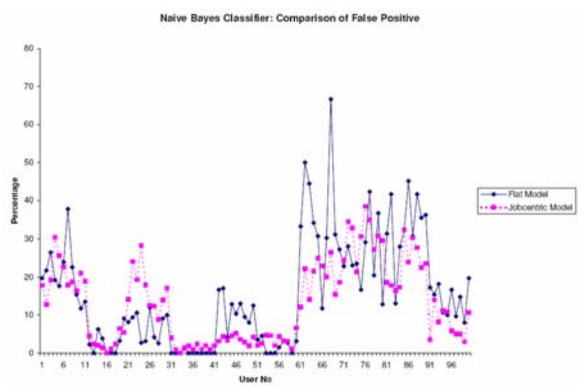
(a) Markov Model - True Positive Rate



(b) Markov Model - False Positive Rate



(c) Naive Bayes Model - True Positive Rate



(d) Naive Bayes Model - False Positive Rate

Figure 11: Evaluation of Flat Text-Based Approach vs. Job-Centric Approach

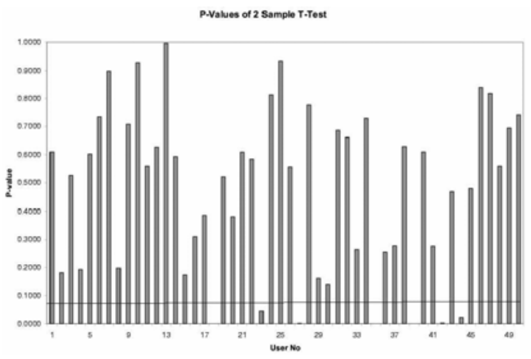
4.3 RACOON Evaluation

Ideally, it is desired to evaluate both data generation paths of RACOON. However, since there is no mechanism through which open-ended data sets can be benchmarked, only Path II is evaluated using Schonlau's data set. For the evaluation process, various metrics were used, and the results obtained provided good insight into the quality of data generated by RACOON.

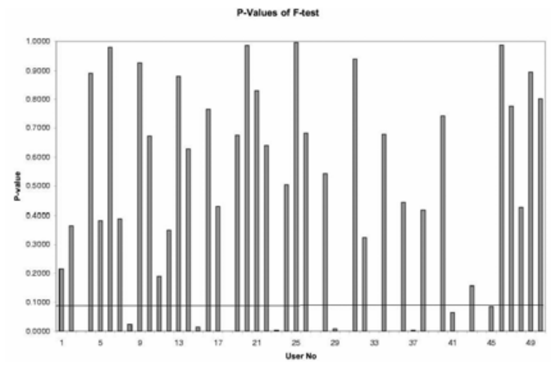
4.3.1 Statistical Similarity Measures

Commonly used statistical similarity measures take two samples and verify whether they came from the same population. The underlying hypothesis testing framework requires that for strong similarity, both samples should lie in the 99% (or more) confidence interval, or in other words display a high degree of overlap. The results of four statistical measures are shown, viz., 2 sample t-test, F-test, Levene's test and Mann-Whitney test.

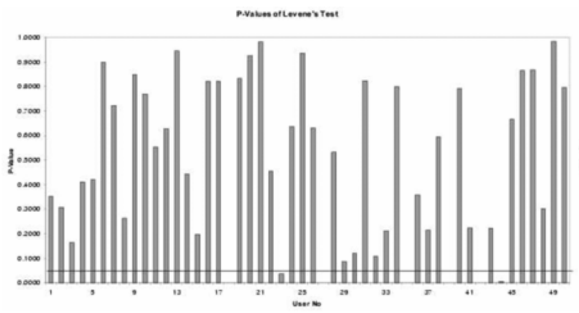
Figure 12 shows the *p*-values (indicated by the vertical bars) obtained for both actual and generated data sets for each test. A very high confidence level of 99% was used for the statistical tests and a *p*-value of at least 0.05 indicates good evidence that both data sets are statistically similar. In a typical run on which the tests were conducted, 7 users failed the 2 sample t-test, 14 users failed the F-test, 13 users failed the Levene's test and only 2 users failed the Mann-Whitney test. In order to understand the reason for these failures, the histogram plots of failed users was visually inspected. Figure 13 shows the comparison of actual data set and generated data set of one such user. The two histograms are identical except for an outlier spike (circled) which effectively shifted the moments towards it and outside the 99% confidence interval. This spike can be traced to RACOON's data generation routine which on occasion repeatedly generates the same command, although it didn't occur with high frequency in the actual data set.



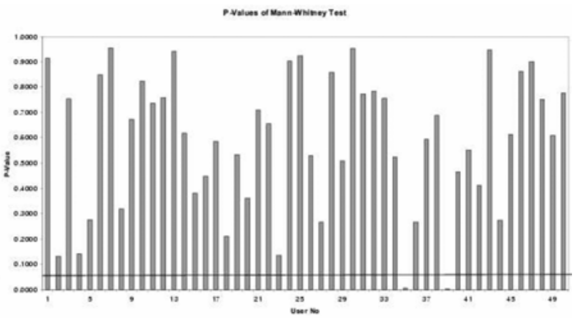
(a) p-value for 2 sample t-test



(b) p-value for F-test



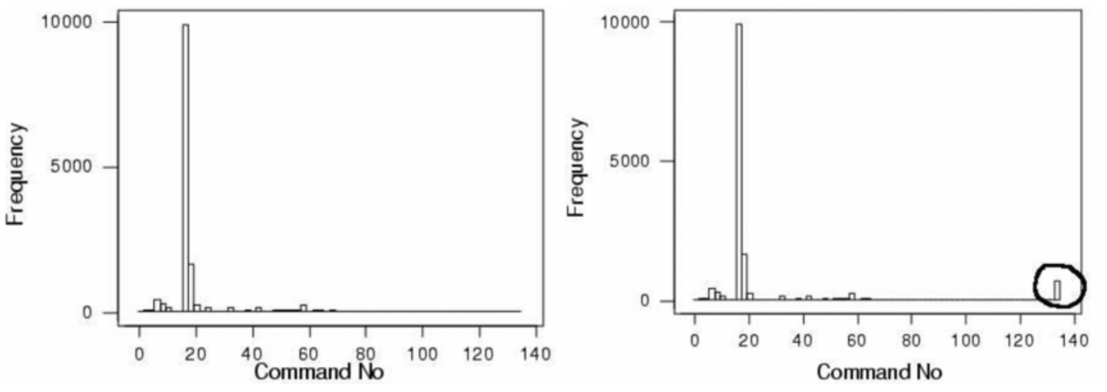
(c) p-value for Levene's test



(d) p-value for Mann-Whitney test

Figure 12: Statistical similarity tests

Plot of *p*-value for each pair of users from actual and generated data set (a) 2 sample t-test, (b) F-test, (c) Levene's test, and (d) Mann Whitney test



(a) Actual data (b) Generated data

Figure 13: Comparison of histograms of user commands for a failed user

4.3.2 Information Theoretical Measures

Researchers have shown that information theoretical measures such as entropy and relative entropy can be used for anomaly detection. The main idea is that two similar data sets will have the same amount of randomness. We have separately computed the entropy values of both actual and generated data sets for all 50 users in a typical run and this is shown in Figure 14 (adjacent 100 vertical bars). It can be seen that both the data sets have very similar entropy values, differing only by 1.5 to 2.0 for almost all users.

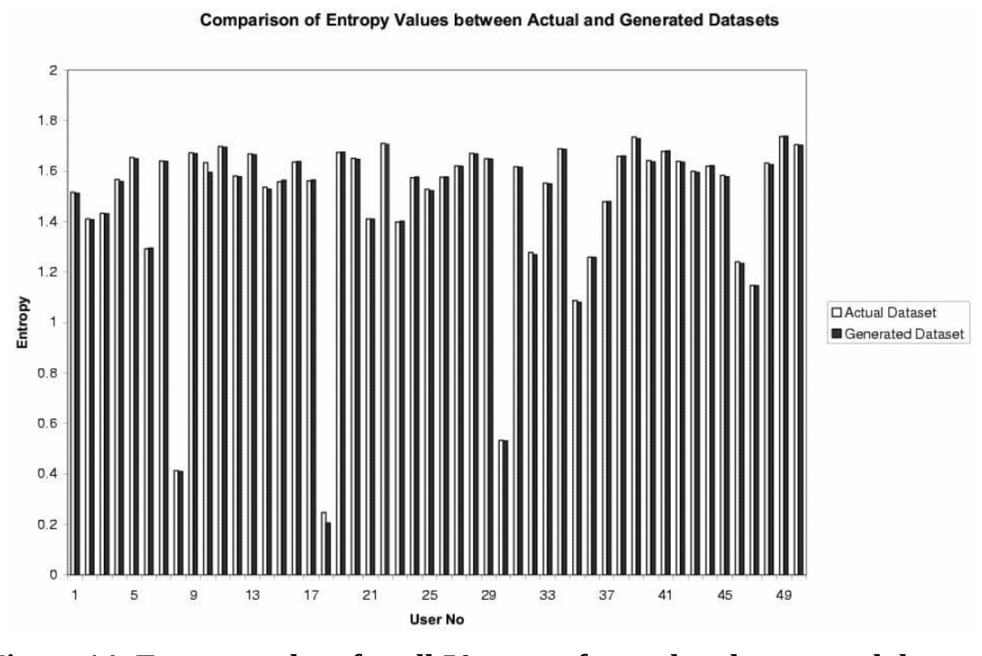


Figure 14: Entropy values for all 50 users of actual and generated data set

4.3.3 Anomaly Detection Algorithms

In our experiments, both the statistical and information theoretical measures were generally in favor of strong similarity. However, both these classes of tests are dependent on command frequencies and the first few moments. Therefore, for a true evaluation, the generated data was also compared against actual data using algorithms which have been shown to perform good masquerade detection; in particular, Uniqueness by Schonlau et al. [8] and Naive Bayes Classifier by Maxion et al. [18]. Both these algorithms reportedly perform very well on Schonlau's data set with high detection rates or low false positive rates or both. We considered these two anomaly detection algorithms and ran their implementations on the actual and generated data sets. Figure 15 shows the true and false positive rates obtained when a user is masqueraded with the remaining 49 users. We have chosen a user who is a representative of both the best case and worst case scenario for the RACOON generated data set. In general, the true positive and false positive rates for the actual and generated data set follow each other closely. However, there are cases where drastic gaps can be seen. This is mainly because RACOON's data generation parameters do not dynamically adapt to the actual data; some users are extremely noisy, while others are not. Since RACOON uses the same procedure for all users, these cases stand out. This shows that RACOON is not perfect and we are looking at ways to enable RACOON to handle diverse cases differently. Note that in the figures, connected lines have been used because it is a convenient way to show the gap between the results obtained in the two data sets, and they are not meant to indicate any correlation behavior.

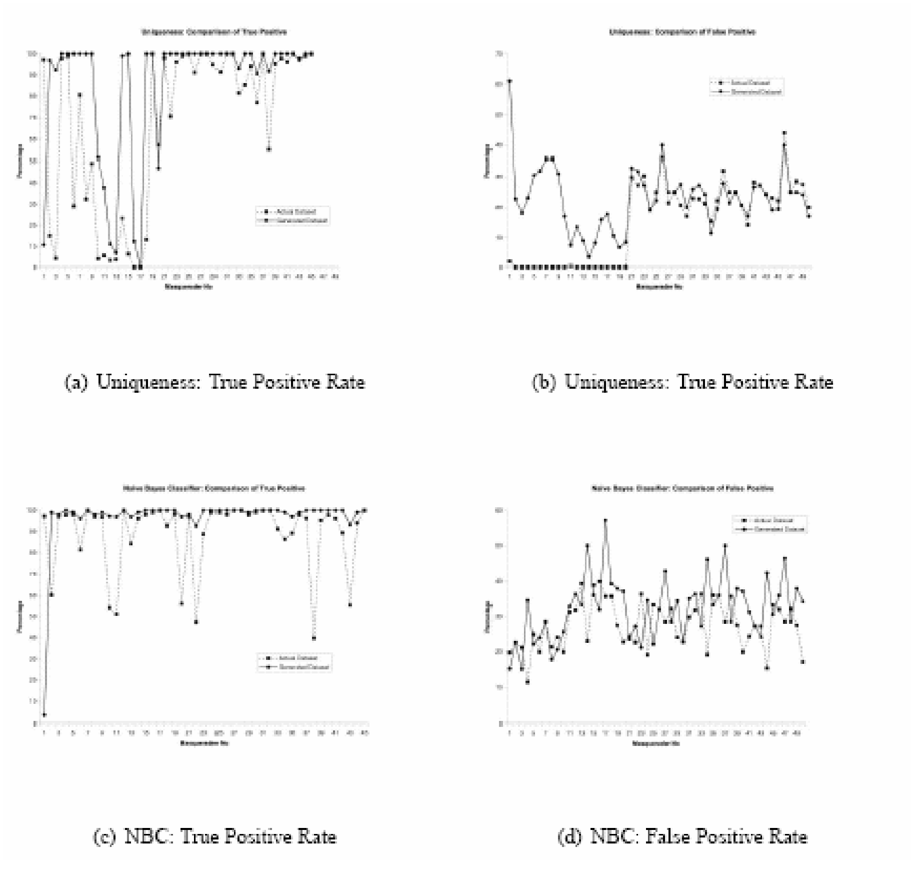


Figure 15: True positive & false positive rates 1v49 for both actual and generated data sets

(a) Uniqueness true positive rate, (b) Uniqueness false positive rate, (c) Naive Bayes Classifier true positive rate, (d) Naive Bayes Classifier false positive rate

4.4 Intrusion Detection in GUI Based Systems

We present a new framework for creating a unique feature set for user behavior on GUI based systems. We have collected real user behavior data from live systems and extracted parameters to construct these feature vectors. These vectors contain user information such as mouse speed, distance, angles and amount of clicks during a user session. We model our technique of user identification and masquerade detection as a binary classification problem and use a Support Vector Machine (SVM) to learn and classify these feature vectors. We show that our technique can provide detection rates of up to 96% with few false positives based on these feature vectors. We have tested our technique with various feature vector parameters and conclude that these feature vectors can provide unique and comprehensive user behavior information and are powerful enough to detect masqueraders.

We now elaborate on our experimental methodology which includes data collection, parameter extraction and feature vector construction.

4.4.1 Data Collection

We collected real user behavior data for multiple users and extracted unique parameters to be able to construct the feature vectors. For this purpose, we developed an active system logger using Microsoft .NET framework and C# language on the Windows XP system. We chose to use .NET framework due to its ease of use and ability to interact with various Windows components seamlessly. This logger is designed such that it is able to collect system events due to all possible user activities on the system in real-time. The logger collects events such as keyboard activity, mouse movement coordinates and mouse clicks, system background processes and user-run commands. The logger collects the timestamp information along with the events, such as mouse coordinates, mouse clicks, process information and keyboard statistics. Because of the fine granularity of the CPU ticks and the fact that we are logging the mouse coordinates, the data collected through the logger can be huge and thus in the next two subsections we show how we extract useful information from this data and construct feature vectors for use with SVM. We sanitized the logged data to remove any personally identifiable information and retained only the behavioral information such as mouse activities.

Feature Extraction. After collecting the real user data, we need to find useful parameters of the user behavior to be able to construct a unique feature vector for training and testing with SVM. We developed a feature extraction engine to parse the logged data. We extracted the following features:

- Mouse clicks (lc and rc): the average number of left and right mouse clicks per user session as well as activity for each 10 minute window during the session.
- Distance (d): the average distance traveled by mouse per event. Examples of such events are closure of a window, start of a process and process termination.
- Mouse speed (s): the average speed of movement for the entire session as well as for events.
- Mouse movement angle per event (θ values): the angles of mouse movement relative to the x-axis for each event.

We calculated the angles θsi and θtj for starting position of mouse for an event and the end position or a click respectively. Due to the fact that these events can be reached at from either left or right directions, there is a total of 4 angles (2 each for left and right), namely θsl, θtl, θsr and θtr. We now compute the total number of features to construct the feature vector.

Calculation of Features. We constructed 16 features after extracting parameters as described in previous sections. These are:

- Mouse clicks: (lc + rc) = 2
- Distance: d = 1
- Mouse Speed: s = 1
- Angles: (θsl + θtl + θsr + θtr) = 4

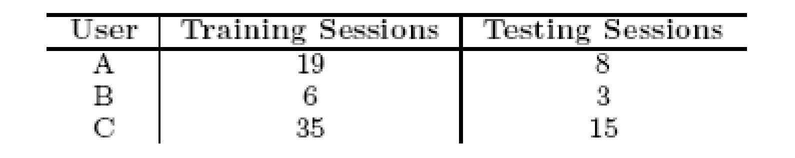
The total of these raw features is 8. Additionally, we calculated the mean m and standard deviation sd for all the raw features above. This gives a total of 16 unique features represented as:
(lc; rc; d; s; θsl; θtl; θsr; θtr) * (m; sd)

Although we limited our experiments to these 16 basic features, one could consider more unique features which may give a better classification. This will be part of the future work. For our experiments, we calculated the mouse clicks (lc and rc) per 10 minutes of activity for a sliding window starting from the first instance as follows:

> If the session starting time t = 31, then we calculate the clicks for 10 minute period following 31 (first period would be [31-41] and second will be [41-51] and so on). After finishing this iteration, we slide the window to time t = 32 and calculate the clicks for the following period such that the first period would be the range [32-42], second will be [42-52] and so on. We calculate the values for 10 iterations, after that the pattern repeats itself. We take this approach to generate the features because of the use of the support vector machine algorithm for learning and classifying behavior features.

4.4.2 Experimental Results and Discussion
We collected GUI based datasets in our lab from 3 users. This data was fed to the parsing engine to obtain 16 features. These features were used to create tuples for feeding to the SVM. The methodology to train and test the system was as follows:

- Datasets were obtained for three distinct users A (27 sessions), B (9 sessions) and C (50 sessions).
- The obtained data set was split for training and testing as follows:



User	Training Sessions	Testing Sessions
A	19	8
B	6	3
C	35	15

- Used the window function as described in previous section and create 160 tuples from the 16 features.
- Used training and testing sets for SVMs as described above and calculated the detection rates and false positives.

In order to run our data tuples we used SVMlight [19] software, which implements Vapnik's Support Vector Machine [20]. We calculated classification results and detection rates for three users. The results from this test are shown in Table 2.

Table 2: Detection Rates

Users A, B and C with 600 Training and 260 Testing Samples for 16 Features; C = correctly classified, IC = incorrectly classified

Model	Detection Rate	False Positives	SVs
A - B, C	92.31% / 240 (C)	20 (IC)	308
B - A, C	85.77% / 223 (C)	37 (IC)	86
C - A, B	96.15% / 250 (C)	10 (IC)	166

We also obtained classification results by varying the number of features from 16 down to 14, 12 and 8 and determining the change in the detection rates. This helps us in determining the optimal number of features for best detection rates. We tabulate the features corresponding to each set in Table 3.

Table 3: Calculation of Features for Different Sizes Features

Features	Description
16	$\{(lc + rc + d + s + \theta_{sl} + \theta_{tl} + \theta_{sr} + \theta_{tr}) * (m + sd)\}$
14	$\{(d + s + \theta_{sl} + \theta_{tl} + \theta_{sr} + \theta_{tr}) * (m + sd) + (lc + rc)\}$
12	$\{(\theta_{sl}, \theta_{tl}, \theta_{sr}, \theta_{tr}) * (m + sd) + (lc + rc + d + s)\}$
8	$\{(lc + rc + d + s + \theta_{sl} + \theta_{tl} + \theta_{sr} + \theta_{tr}) * (m)\}$

Table 4: Detection Rates Obtained by Varying the Number of Features for User C

Features	Detection Rate	False Positives
16	96.15%5	37
14	74.23%	67
12	73.85%	68
8	73.85%	68

As can be seen from Table 4, the detection rate goes down as we decrease the number of features from our set. Although some earlier approaches used more features to profile the user [21], we can observe from our results that 16 is a good number of features to make a distinction between multiple users. Also, incorporating more features in the set, such as keystroke dynamics and process information, will only marginally improve the results. The amount of effort for logging additional data, parsing it to extract feature information will subdue the marginal improvement in detection rate. We would like to point out the fact that the results presented are preliminary and based on a small set of users (three). It is possible to construct feature vectors for more users and test the system for its accuracy.

5 Conclusions

In this project, the focus of investigation has been user-level intrusion detection. Key issues have been studied and important problems plaguing user-level intrusion detection have been identified. The primary goal has been to address these immediate problems and lay the basic foundation for further studies and improvements. The main results of this dissertation are summarized as follows.

Job-Centric Model. Although it is well-known that user command behavior has a *cause-effect* relationship, that is, a user executes commands to complete a particular task, there has been no known effort which tries to capture this aspect well. A job-centric model has been proposed and validated wherein it is argued that users have a high-level task in mind and require one or more functionalities to accomplish this task. In our model, this high-level goal is called a *job,* the functionalities are called *meta-commands* and actual instantiations are called *commands*. User profile is a combination of all these factors and not merely a first-order statistical inference from user command trails.

Rule-Based Learning. Anomaly detection by nature is a decision-making process under partial or incomplete information. Therefore, the nature and amount of information that is captured is a critical factor. By involving the user in the security process, we gain on several grounds. First, the user is conscious of the security system in place and legitimate users tend to cooperate with the IDS. Second, the IDS learns based on the rules specified by the user during the session-scope query. This greatly reduces the need to make guesses about the same information and hence, the corresponding errors reduce significantly. At the same time, an intruder who is not aware of the user's profile is more likely to commit errors which appear as substantial deviations from the legitimate user's profile.

Sequential Hypothesis Testing. Intrusion detection can either be after-the-fact or proactive. The latter goal is more desirable but it also entails an online detection mechanism. Online algorithms for user-level intrusion detection are already known but they are cumbersome to use mainly because thresholds for detection have to constantly tweaked to obtain near-optimal detection performance. *Sequential hypothesis testing* is used because it solves both these issues simultaneously.

Data Collection and Generation. Collection of user command trails is a highly debatable issue with arguments for and against it. Data is required to train the IDS, but at the same time there are concerns regarding user privacy. This is a very difficult problem to solve and therefore, user-level IDS algorithms continue to use existing datasets which are known to be flawed. A user command data generation mechanism is proposed by leveraging our job-centric approach and show that realistic data can be generated. This solves an important problem facing user-level IDS evaluation.

Implementation. Based on the ideas which have been proposed, we have implemented a preliminary prototype called DRUID over Linux and FreeBSD using ptrace(2) for the online user command monitor. GUI related modules have been implemented through a combination of X and gtk libraries. The entire implementation is written using the C programming language. The code runs exclusively in user-space, hence, becomes highly portable and requiring no changes to the underlying kernel. This implementation is still research-grade and there are several known bugs which have to be fixed.

At the later stages of the project, we attempted to port DRUID to a Windows platform. We have developed a modeling and simulation framework to address multistage attacks in GUI systems and presented some preliminary results on masquerade detection using GUI feature sets.

Two novel concepts, namely, proactive detection and recovery using attack trees and QoS throttling to elicit user cooperation for enhanced security have been developed to make DRUID robust and useful in the field. This part of the research will be an ongoing effort beyond the duration of this project.

6 Publications and Patents

Publications under this grant include:

1. Upadhyaya S.J., and K. Kwiat, “A distributed concurrent intrusion detection scheme based on Assertions”, *1999 SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems*, Chicago, IL, pp. 369-376, July 1999.

2. Upadhyaya S., “Attack recognition in distributed systems by assertion checking”, poster presentation, *Information Institute-SAB reception*, Air Force Research Laboratory, Information Directorate, Rome, NY, Dec. 6, 1999.

3. Mantha K., R. Chinchani, S.J. Upadhyaya and K. Kwiat, “Simulation of intrusion detection in distributed systems”, *SCS Summer Simulation Conference*, Vancouver, Canada, July 2000.

4. Upadhyaya S., R. Chinchani and K. Kwiat, “A comprehensive reasoning framework for information survivability”, *2nd Annual IEEE Systems, Man, and Cybernetics Information Assurance Workshop*, West Point, NY, pp. 148-155, June 2001.

5. Upadhyaya S., R. Chinchani and K. Kwiat, “An analytical framework for reasoning about intrusions”, *IEEE Symposium on Reliable Distributed Systems*, New Orleans, LA, pp. 99-108, October 2001.

6. Upadhyaya S., “Attack recognition and shielding in distributed information systems”, poster presentation, Information *Institute-SAB reception*, Air Force Research Laboratory, Information Directorate, Rome, NY, Nov. 5, 2001.

7. Chinchani R., S. Upadhyaya and K. Kwiat, “Towards the scalable implementation of a user level anomaly detection system”, *IEEE MILCOM 2002*, Anaheim, CA, October 2002.

8. Upadhyaya S., “A Tamper-Resistant Framework for Unambiguous Detection of Attacks in User Space Using Process Monitors”, *1st New York State Cyber Security Conference*, Utica, NY, Feb. 2003 (poster).

9. Chinchani R., S. Upadhyaya and K. Kwiat, “A Tamper-Resistant Framework for Unambiguous Detection of Attacks in User Space Using Process Monitors”, *IEEE International Workshop on Information Assurance*, Darmstadt, Germany, pp. 25-34, March 2003.

10. Upadhyaya S., R. Chinchani and K. Kwiat, “New methods for attack detection”, *AFRL Information Institute Workshop*, Rome, NY, June 2003 (poster).

11. Garg, A. Shambhu Upadhyaya, Ramkumar Chinchani, Kevin Kwiat, SIMS: A Modeling and Simulation Platform for Intrusion Monitoring/Detection Systems", *Summer Computer Simulation Conference 2003*, Montreal, Canada, July 2003.

12. Chinchani R., A. Muthukrishnan, M. Chandrasekaran and S. Upadhyaya, "RACOON: Rapidly Generating User Command Data for Anomaly Detection from Customizable Templates", *20th Annual Computer Security Applications Conference*, *Tucson, AZ,* December 6-10, 2004.

13. Upadhyaya S., K. Kwiat, R. Chinchani and K. Mantha, "Encapsulation of Owner's Intent – A New Proactive Intrusion Assessment Paradigm", in Vipin Kumar, Jaideep Srivastava and Aleksandar Lazarevic, Eds., *Managing Cyber Threats: Issues, Approaches and Challenges*, Springer, pp. 221-245, 2005.

14. Ray, Indrajit and Nayot Poolsappasit, "Using Attack Trees to Identify Malicious Attacks from Authorized Insiders", Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS 2005), September 2005.

15. Garg A., V. Sankaranarayanan, S. Upadhyaya and K. Kwiat, "USim: A User Behavior Simulation Framework for Training and Testing IDSes in GUI Based Systems", *39th Annual Simulation Symposium*, Huntsville, AL, April 2006.

16. Sankaranarayanan V. and S. Upadhyaya, "A Trust Assignment Model based on Alternate Actions Payoff", *4th International Conference on Trust Management, Pisa, Italy, May 2006.*

17. Garg A., Ragini Rahalkar, Shambhu Upadhyaya and Kevin Kwiat, "Profiling Users in GUI Based Systems Masquerade Detection", *to appear in 7th IEEE Information Assurance Workshop,* West point, NY, June 2006.

No patents were filed under this project.

We have benefited from fruitful discussions with Program Manager Kevin Kwiat who visited Buffalo several times during the course of this project.

7 Ph.D. Theses Supported by the Project

Ramkumar Chinchani, 2005
Ashish Garg (to defend in August 2006)
Vidyaraman Sankaranarayanan (to defend in 2007)

References

[1] M. Namjoo, “Techniques for Concurrent Testing of VLSI Processor Operation”, *Proc. International Test Conference*, 1982, pp. 461-468.

[2] S. Upadhyaya, R. Chinchani and K. Kwiat, “An analytical framework for reasoning about intrusions", *20th IEEE Symposium on Reliable Distributed Systems*, 2001, pp. 99-108.

[3] D. Ferraiolo and R. Kuhn, “Role Based Access Control”, 15th National Computer Security Conference, 1992.

[4] S. Upadhyaya, K. Kwiat, R. Chinchani and K. Mantha, “Encapsulation of Owner's Intent – A New Proactive Intrusion Assessment Paradigm”, in Vipin Kumar, Jaideep Srivastava and Aleksandar Lazarevic, Eds., *Managing Cyber Threats: Issues, Approaches and Challenges*, Springer, pp. 221-245, 2005.

[5] A. Wald, *Sequential analysis*, J. Wiley and Sons, New York, 1947.

[6] D. Johnson. Sequential hypothesis testing, http://cnx.rice.edu/content/m11242/latest/?format=pdf, 2003.

[7] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, “Fast portscan detection using sequential hypothesis testing”, *IEEE Symposium on Security and Privacy*, May 2004.

[8] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, “Computer intrusion: detecting masquerades”, *Statistical Science*, 16(1):58–74, 2000.

[9] Ramkumar Chinchani, “A Job-Centric Approach To User-Level Intrusion Detection”, *Ph.D. Dissertation*, Dept. of Computer Science and Eng., University at Buffalo, 2005.

[10] T. Lane, “Purdue UNIX User Data”, http://www.cs.unm.edu/terran/research/data/Purdue_UNIX_user_data.tar.gz, 1999.

[11] T. Lane and C. E. Brodley, “Sequence matching and learning in anomaly detection for computer security”, *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[12] M. Schonlau, “Masquerading User Data”, http://www.schonlau.net/intrusion.html, 1998.

[13] Ramkumar Chinchani, A. Muthukrishnan, M. Chandrasekharan and Shambhu Upadhyaya, "RACOON: Rapidly Generating User Command Data For Anomaly Detection From Customizable Templates", A*nnual Computer Security Applications Conference* (ACSAC), 2004.

[14] S. J. Bates., “User Behavior in An Interactive Computer System”, *IBM Systems Journal*, 13:1.18, 1974.

[15] Garg A., V. Sankaranarayanan, S. Upadhyaya and K. Kwiat, “USim: A User Behavior Simulation Framework for Training and Testing IDSes in GUI Based Systems”, *39th Annual Simulation Symposium*, Huntsville, AL, April 2006.

[16] Indrajit Ray and Nayot Poolsappasit, “Using Attack Trees to Identify Malicious Attacks from Authorized Insiders”, Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS 2005), September 2005.

[17] D. Wagner and P. Soto, “Mimicry attacks on host-based intrusion detection systems”, ACM CCS, 2002.

[18] R. A. Maxion and T. N. Townsend, “Masquerade detection using truncated command lines”, *International Conference on Dependable Systems and Networks (DSN’02),* pages 219–228, June 2002.

[19] T. Joachims, “SVM light: Support Vector Machine," 2004, http://www.cs.cornell.edu/People/tj/svm light/index.html.

[20] V. N. Vapnik, “The Nature of Statistical Learning Theory”, Springer, 1995.

[21] M. Pusara and C. E. Brodley, “User re-authentication via mouse movements”, *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, (Washington DC, USA), pp. 1-8, 2004.