Towards the Scalable Implementation of a User Level Anomaly Detection System

Ramkumar Chinchani, Shambhu Upadhyaya and Kevin Kwiat

Abstract— **Traditional intrusion detection systems can be broadly classified as misuse and anomaly detectors. Misuse detectors attempt detection by matching the current system/user activity against known signatures and patterns. As opposed to this, anomaly detection works by developing a reference graph and comparing the ongoing activity against it. Any significant deviation is flagged as an intrusion. Anomaly detection is more promising because of its potential to detect unseen types of attacks. However, both techniques have conventionally relied on audit trails sampled deep inside the system via probes and the sheer size of the data allows only after-the-fact and off line detection. In recent past, there have been efforts to capture the semantics of system activity for more rapid detection and this can typically be done at levels closer to the user. In our earlier works related to this effort, we presented a scheme and a reasoning framework to detect intrusions based on the encapsulated user intent. This paper addresses the scalability and implementation aspects of the system by introducing concepts such as workspaces and meta-operations. Although this security system is a general anomaly detection system, it is amenable to operator fault recovery. While encryption provides secure communication channels, it leaves the end points exposed. Our security system has the additional capability of handling insider attacks relevant in this context.**

Keywords— **Anomaly Detection, Scalability, User Level Detection**

I. (U) Introduction

A large number of intrusion detection systems have been deployed with varying success in order to address system security, an issue that is becoming an ever-growing concern. Known taxonomy [1] classifies these intrusion detection systems into a wide variety of categories. One popular classification divides these into misuse and anomaly detection systems. They are alternatively known as knowledge-based and behavior-based intrusion detection systems respectively.

Misuse detectors [2], [3] can detect and identify known intrusions with a high accuracy. They achieve this by maintaining a knowledge base of signatures corresponding to the intrusion scenarios and then comparing the actual system activity obtained from audit trails against this knowledge base. A positive match signals an intrusion with the relevant information about it. Some of the techniques used are colored Petri Nets [4], state-transition analysis [5], [6] and property-oriented analysis [7]. Since the general approach is to look for specific patterns or information, misuse detection is relatively more accurate. Attacks can be distinguished and given distinct names. However it suffers from the limitation that it cannot detect unseen types of attacks.

R. Chinchani is a PhD student in the Dept. of Computer Science and Engineering at State University of New York at Buffalo, USA. Email: rc27@cse.buffalo.edu

S. Upadhyaya is an Associate Professor in the Dept. of Computer Science and Engineering at State University of New York at Buffalo, USA. Email: shambhu@cse.buffalo.edu

K. Kwiat is with the Air Force Research Laboratory, Rome, NY, USA. Email: kwiatk@rl.af.mil

Anomaly detection [8], [9] is the alternative approach. Normal system activity is observed and modeled, and this serves as a reference. In this context, intrusions are often defined as "noise" or anomalies that deviate from this reference graph. As a consequence, this technique can potentially detect unseen types of attacks and is therefore complete. While this approach has its apparent advantage, it suffers from some limitations. An anomaly detection system has to undergo long training sessions of noiseless data to construct fairly accurate statistics. It is also very hard to set appropriate thresholds that bind all scenarios. Due to the inherent nature of the approach, nomenclature of attacks is not relevant. In spite of these limitations, considerable efforts have been invested in this direction to achieve the elusive completeness property of this technique.

An ideal intrusion detection system is a tough goal to achieve. It has these desirable characteristics.

- Low latency of detection by rapid decision-making
- Low false positive/negative rate
- Stronger deterrence to cracker's attacks by active/on-line monitoring
- Scalable to large environments
- Ability to be deployed in a heterogeneous and distributed environment

A careful scrutiny of the above goals reveals a close interdependence between each other. An intrusion detection system that can process lesser information is able to respond faster, make fewer mistakes and scale well due to the lower overheads it generates.

The disparity of accuracy and completeness apart, both misuse and anomaly detection systems rely on huge amounts of audit data samples. Also, both require and maintain permanent and transient data to enable them to make decisions regarding intrusions. This typically departs from the goals mentioned above.

From the perspective of our work, we discuss some of the relevant approaches in contemporary anomaly detection. In recent past there has been considerable work done in the area of behavior-based intrusion detection and novel approaches have been proposed. Cost-based models [13], [14] attempt to detect intrusions by using cost as a metric. The system activity, resource usage and intrusion damage are quantified by assigning costs to them and decisions regarding the actual detection and recovery are based on them. User intent modeling [15] attempts to infer the user's intent based on low level system activity and preempts any intrusions if it is established or believed that the user's intent is malicious. Intruder identifica-

tion by monitoring at the system command level by the use of Markov Chains [16], [17] has been explored to model user behavior. However, these techniques suffer from certain limitations. These approaches require extensive data mining and the use of machine learning algorithms to extract the useful features for decision-making. This increases the latency of detection and processing overhead.

Our approach, i.e., intrusion detection based on user intent encapsulation [18] is a novel approach where the user's intent is procured by directly querying the user irrespective of whether he is a bona fide user or an intruder, and the outcome is compared to the expressed intent. This is a proactive and more aggressive methodology [19] and a significant departure from the traditional schools of thought. As pointed out earlier, most approaches use very low level audit data for analysis. System activity is more or less generated by the users using the system and capturing the semantics cannot be done efficiently at such low levels. A comprehensive cost-based reasoning framework [20] has been developed where the user activity is quantified based on costs attributed to the usage of resources and the amount of deviation from the known activity, and a decision regarding an intrusion is made. Advantages of this approach are a smaller search space and reduced false positives. On the downside, since monitoring is done at the user level, this scheme offers little in terms of detecting low-level network attacks and external denial-of-service attacks.

While encryption offers secure communication channels, the endpoints of communication are exposed. For example, this is the case in Virtual Private Networks (VPN). Most system compromises occur at the user level and hence require stronger protection at that level. Insider attacks involving *social engineering* are not uncommon. Our technique has the additional capability of handling such attacks.

The main focus of this paper is to propose enhancements to this approach by addressing outstanding issues regarding scalability and describing related architectural changes prior to full-fledged system implementation. We introduce concepts like prioritized workspaces and meta-operations to tackle these issues.

A. Paper Organization

Section 2 gives a brief description of the work that is already done and the problems that we are addressing in this paper. Section 3 presents the proposed solutions. Sections 4 and 5 describe the systemic modifications and implementation issues. Since the prototype is not fully functional at this point, only preliminary results are presented in Section 6. Discussions and future work are presented in sections 7 and 8.

II. (U) Background

For the sake of completeness, this section briefly discusses our prior work. At the end of this section, we present the problems of interest.

When a user wants to perform some jobs on a system, he is interactively queried about his intent and this procured intent forms his *session scope* [18]. Since user interaction is an issue, the query is performed through a graphical interface in a controlled manner [21]. Obvious questions such as "How can the user express his intent?", "What if the user is lying?", "What if an intruder masquerades as a user?" and so on have been adequately addressed in [18]. Upon setting his approximate schedule, the user is allowed into the system where he begins performing his tasks. Each user is identifiable by his user-id and the operations he performs. This distinction is reflected by the user's profile. As this system is an anomaly detection system, intrusions are viewed as significant deviations from this profile. Development of user profiles and reasoning about intrusions based on such profiles has been elaborately discussed in [20]. Cost has been used as a metric for the purposes of decision-making. The cost of every job is expressed as a linear sum of two components, the cost of an operation and the cost of sequence of operations. Another interesting aspect of that work was the use of dynamic thresholds to reduce false positives.

Given a set of operations, the different partial orders or permutations can define different jobs. In order to compare the user's intent with the outcome rapidly, it is essential to unambiguously establish the relationship between the operations and the jobs that a user is performing. We illustrate this with a simple example. Consider a set of operations $O = \{A, B, C, D\}$. Consider the jobs J_1, J_2, J_3, etc., defined as a partial order on the set O as $J_1 = ABC$, $J_2 = BC$, $J_3 = ABCD$ and so on. Some of these jobs are malicious and others are not. Given a workspace to perform his jobs, the user can begin with any job with the corresponding operations. If he executes the operation A, it is not possible to infer whether he is performing job J_1 or J_3. Now if the user continues and performs an operation B, it is unclear whether jobs J_1 or J_3 are being continued or a new job J_2 is being started. We demonstrate the complexity of the problem as follows. Let there be n operations. Various jobs can be performed by choosing a subset of these operations and defining an ordering on these subsets. Without repetition of operations, these ordered sets represent different permutations on the set O.

$$\text{Total number of permutations} = \sum_{k=1}^{n} P(n,k) \tag{1}$$

$$\text{Now, } \sum_{k=1}^{n} C(n,k) = 2^n - 1 \tag{2}$$

$$\text{and } \sum_{k=1}^{n} P(n,k) > \sum_{k=1}^{n} C(n,k) \tag{3}$$

$$\text{Hence, } \sum_{k=1}^{n} P(n,k) > 2^n - 1 \tag{4}$$

The computational complexity to resolve these ambiguities in the worst case is poorer than exponential. However, in reality we may not see such worst case scenarios and hence all possible orderings need not be considered. All the same, this shows how hard the problem is. Even a small increase in the

number of operations results in a very large increase in computation. On the other hand, the good news is that a decrease in the number of operations causes a corresponding exponential reduction in computation.

By actively querying the user for his intent, the search space for jobs and the associated operations is reduced and we have a more focused reference line for comparison. But given one workspace to perform the jobs, the problem as such is unchanged even though its size is now a little smaller. Therefore, at this point, although we have a framework for reasoning about anomalous intrusions, it does not scale well. Decision-making is preceded by the overhead due to the ambiguity resolution. The problem we now face is - Is it possible to do any better? It must be noted that no known polynomial order solution exists for this problem.

Our efforts in this paper are directed in addressing this problem and presenting the alterations in design and implementation. We show that the size of the problem can be reduced further using the *divide and conquer* approach and this also eliminates the need for ambiguity resolution thereby drastically reducing the processing overhead.

III. (U) The Engineering Approach

In order to use the divide and conquer technique successfully, one must identify the sub-problems and partition the main problem into these sub-problems. Some aspects of the original scheme lend themselves to such optimizations. We will evolve some conceptual machinery and eventually delve into the actual engineering changes.

A. Workspaces

The session scope can be viewed as a function that partitions the entire set of operations into smaller subsets each associated with a job. By choosing a few jobs during the intent query, the user implicitly creates these partitions. However, the distinctness of these partitions is lost when the user begins to work in a given workspace or environment. A *workspace* is an environment allocated to a particular job. It serves as a bracket of privileges for that job. Multiple workspaces can be defined; one for each job that is required to be performed. Figure 1 elucidates this transformation. It must be noted that the use of workspaces doesn't occlude common operations among jobs. It merely implies that each job is being performed in a workspace assigned to it regardless of other jobs being performed outside that workspace.

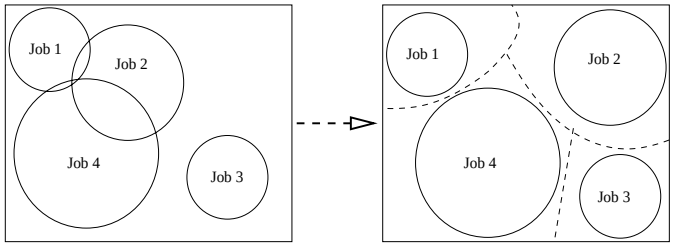

Fig. 1. **Jobs being assigned separate workspaces.**

This workspace protocol can be implemented in two ways:

- **Physical workspaces**

On completion of the user's intent expression, one workspace is allocated to each chosen job. The user is now required to perform operations related to a job in the workspace allocated to that job, which is done explicitly. Violation of this restriction leads to signaling of anomalous activity. This implementation makes rigid and visible compartments for all the jobs to be performed. This setup raises obvious concerns about how realistic this would be and would the user be subject to more inconvenience by imposing such a requirement. The concept of workspaces by itself is not new. In several Unix systems, the user interacts with a window manager when working in a graphical environment. Window managers like *ctwm* [22], *fvwm* [23], *CDE* [24], etc., allow the user to define virtual screens in which the user can launch and place the various X clients. Workspaces are a very common feature among these modern window managers; the main reason being that they allow the user to organize the work and the windows better, avoiding unnecessary clutter on the desktop.

- **Virtual workspaces**

In this implementation, the mapping of the workspaces is done using menus. Each job has a corresponding menu item and commands can be launched off from the appropriate menu items. In this case, the workspaces are implicitly defined and are more transparent to the user. This makes it more convenient to the user because he is now no longer required to tie an operation to a particular physical workspace. Also, this implementation makes workspaces more independent of the window manager enabling easier portability to other graphical environments where workspaces are not a common feature, e.g., Microsoft Windows doesn't have the concept of multiple workspaces on its desktop hence making it hard to define physical workspaces. It must be noted that the operations are now bound to the menu group they were launched off. All operations created from a particular parent operation belong to the parent's job group.

The use of workspaces is feasible as a consequence of actively obtaining user's intent. The user's explicit intent allows the allocation of workspaces for each job specified. In the absence of such information, it would become largely impractical to use workspaces since one has no concrete knowledge about the jobs that a user wants to accomplish.

Another obvious question follows the above argument, i.e., "What if the user chooses too many jobs?" This problem has a simple and elegant solution. Since the user is indirectly responsible for the monitoring overhead, a proportional factor of the monitoring cost is added to the cost of the job. A third term is added and the cost function of a job [20] is now modified as:

$$Cost(Job) = \alpha \times Cost(Operation) + \beta \times Cost(Sequence) + \gamma \times Cost(Monitoring) \quad (5)$$

A user typically devotes his attention to a few jobs and spends lesser time with other jobs. This makes prioritizing of jobs possible. The ordering of jobs based on priorities gives an approximate reference line to assess deviations in the way the user performs the jobs itself. Also, since we do not expect the user's schedule of jobs to change drastically from day to day, we can use the ordering of the jobs to lay out the user specific data structures in a way that improves access time.

B. Meta-operation

Policy enforcement has to be done at various levels in the system. Given a workspace to complete a job, a user executes various commands. For the same job, different users choose commands based on their preference. This leads to the argument that regardless of the actual command being executed, it is the functionality that matters. This allows for further optimization which we call the *meta-operation.*

A meta-operation is a description of the functionality of a command or operation. It is a function that groups various available commands into sets based on the similarity of their functionality. For example, *EditCommand* is a meta-operation for $\{vi, emacs, xemacs, \cdots\}$.

Since our level of monitoring is at the user command level, we speak of meta-operations at this level. The meta-operation can be thought of as a command descriptor. Consider a *ProgramDevelopmentJob*. It typically involves an editor, compiler and debugger executed in some order. Some commands can perform more than one function, such as *emacs*. Such commands can be represented by different meta-operations. However, based on the jobs being performed, specific functionality can be enforced while occluding others. For example, we can enforce a behavioral rule which allows *emacs* to be used only as an editor during program development and as a mail client in some other job. It is important to ascertain proper runtime behavior for each command. Significant work in terms of specification-based enforcement of security has been done, notably proof-carrying code [25], model-carrying code [26], execution specification [27], etc. Some of the issues are pertinent to our problem and discussed in great detail in these papers. The advantages of this scheme are:

- A large number of commands can be represented by a small number of meta-operations
- Additions and deletions of commands have little effect on policies
- Storage and processing overhead is now restricted to a meta-operation instead of every command

IV. (U) System Architecture

The basic system architecture consists of a host level monitor and several such monitors running on hosts in a network form a hierarchy. Implementation of the afore-mentioned concepts requires modification of the host level monitor while retaining the network level organization. The entire monitoring process (ref. Figure 2) has been divided into three tasks.

- **Command Monitor**

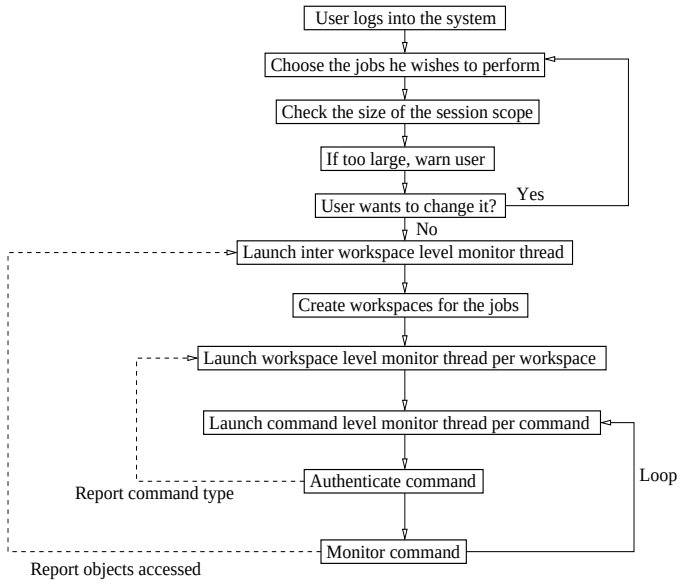

Fig. 2. **Overall monitoring process.**

A command monitor is a thread spawned for every command that is executed. The executed command is then verified and the functionality of meta-operation is enforced. The command can access various objects during its lifetime and requires constant monitoring. It is also responsible for calculating the cost of the operation or command.

- **Workspace Monitor**

As in the case of the command monitor, one workspace monitor is spawned for every workspace. It monitors at the level of sequence of commands, which is at a slightly higher level of monitoring than the command monitor. The workspace monitor receives messages from the command level monitors, and it ensures that the commands are being performed in the known sequences. It is the responsibility of the workspace monitor to calculate the cost of sequence.

- **Inter-workspace Monitor**

It is possible that a user runs commands to form a malicious thread of execution across workspaces. There can be a relation among commands in different workspaces only if they share some common object. Hence, it is necessary to monitor across workspaces. This task is performed by the inter-workspace monitor. One such monitor is created for every user per session.

V. (U) Implementation Issues

Integration of the above concepts and entities into the original framework [21] necessitates consideration of some systemic issues.

- **Process vs. Thread**

Since there are multiple monitors spawned during each user's session, it becomes critical to make a proper trade-off on the nature of these monitors. If the monitor entity is modeled as a process (as was the case in the older implementation), it could

lead to a large outbreak of processes as more users begin accessing the system. Therefore the monitors are now modeled as lightweight threads.

- **User Space vs. Kernel Space**

To capture the semantics of the user's operations, monitoring is done at the highest interface level possible between the user and the kernel, i.e., at the system call level. There has been substantial work on how this can be done with relevance to intrusion detection. There are techniques involving kernel-space interposition [28], user-space interposition [29] and also other hybrid approaches [30], [31]. Each has its own advantages but performance studies show that the overheads are not all that discernible and the choice of any given approach is more of a software engineering issue. Since our monitoring system causes small but non-trivial overhead, we have modeled it largely as a user space system.

- **Object Monitoring**

An intruder typically attempts to access valuable data and hence object monitoring is an integral part of any intrusion detection system. When an object is copied in part or whole into another object, the destination object is marked *dirty* and it is assigned a value equal to the source object. By keeping tabs on the flow of information, the cost of operation of a command involving an object is evaluated, whenever the corresponding process is active.

VI. (U) Preliminary Experiments and Results

The prototype is still in its developmental stages and its exhaustive testing is not plausible at this point. However, we have considered a few test cases to demonstrate the efficacy of the revised techniques discussed in this paper. The comparisons in improvement have been done against the original prototype [21]. Not all criteria lend themselves to impartial evaluation since there have been design and implementation modifications. The original prototype was written in Java while the newer one has been implemented in C++ as a multi-threaded system for the purposes of speed and efficiency. Therefore comparisons of speed and system overhead do not speak intelligently of the gains achieved. The experimental setup simulates an academic environment where users typically perform jobs such as program development, browsing, using mail, etc. Since this system targets anomalies rather than well-known attacks in the misuse detection nomenclature, we find it more useful to analyze the preliminary system in terms of deviations and how rapidly such deviations are detected.

A. Experimental Setup and Evaluation

Currently, the system focuses on host level anomaly detection. Therefore, the experimental system has only one host computer which is a 500MHz Pentium III running Linux (Redhat 7.2, kernel-2.4.7-10). It is installed with the anomaly detection system and users are allowed to perform their jobs on it.

In order to perform anomaly detection successfully, the system has to accrue adequate samples in order to develop the required statistics and stabilize a user's profile. Note that this is done at the user command level as discussed in [20]. For example, the statistics for a *ProgramDevelopmentJob* may read as Command 1: {*emacs* 0.5, *vi* 0.35, *xemacs* 0.10, misc. 0.05}, Command 2: {*gdb* 0.7, misc. 0.3}, etc. Once the command is executed, a command monitor thread tracks the runtime accesses and enforces some behavioral rules for the corresponding meta-operation. For example, *emacs* has to conform to the behavioral rules of the meta-operation *EditCommand* expressed as set of *do* and *don't* statements. The set of do's contains assertions such as {*can open files, can read files, can write files, can close files,* etc.} and the set of dont's contains restrictions such as {*can't use the network,* etc.}.

A limited set of 12 typical jobs were considered and the user activity is monitored. Some sessions were injected with intruder-like activity and the system achieved very high coverage with negligible overheads and false positives. In comparison, the older system gave a much lower coverage for the same test set. Also, there were a large number of false positives and false negatives, and the system overhead was significant.

VII. (U) Discussion

The primary focus of this paper was to introduce and discuss the methodologies that have been adopted in order to achieve rapid anomaly detection without causing substantial overhead on the system. By doing so, one can perform on-line monitoring of a host and respond quicker in terms of intrusion deterrence.

Despite these enhancements, some of the limitations of the older system remain. Since the level of monitoring is at the user level, the new system cannot detect any external low level network attacks such as denial of service attacks.

An interesting aspect of this system is its ability to address insider attacks. For an intruder to be successful in compromising some user account, he must know the user's profile in very specific detail. Another aspect of interest is the system's ability to correct operator faults. In an environment, that is usually free of intrusions and the only anomalies are operator faults, they can be corrected since the interaction level with the system is at the user level.

VIII. (U) Future Work

The final goal of this effort is to create an environment where users do not lose their quality of service and at the same time, enforcement of security occurs on-line. Such a security system would be pertinent to a wide variety of environments such as academics, business, military, etc. In order to achieve that goal, significant efforts need to be invested in our future work and the avenues that require revisiting and revisions are:

- **Information Specification**

Though we have used a simple specification scheme in our test set, it may not suffice in larger scenarios. To make a decision in such environments, maximum information should be captured and represented.

- **Heterogeneous Implementation**

In a realistic deployment scenario, we can expect a wide range of operating systems and the implementation should take such heterogeneity into account.

- **Network Level Monitoring**

The host level anomaly detection has to be extended to a network level system since users can connect to other computers and start a thread of execution on those machines. Communication over networks complicates matters regarding decision-making. Therefore, network level monitoring is a direct extension of the current effort.

(U) ACKNOWLEDGMENTS

This research was supported in part by U.S. Air Force Research Laboratory, Rome, New York, under Contract: F30602-00-10507.

REFERENCES

[1] H. Debar, M. Dacier and A. Wespie "Towards a Taxonomy of Intrusion Detection Systems", *Computer Networks*, Elsevier, Vol. 31, 1999, pp. 805-822.

[2] T. F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst and S. Listgarten, Knowledge based Intrusion Detection", *Proceedings of Annual AI Systems in Government Conference*, Washington D. C., March 1989.

[3] S. E. Smaha, "Tools for Misuse Detection", *Proceedings of ISSA'93*, Crystal City, VA, April 1993.

[4] S. Kumar and E. Spafford, "A Pattern Matching Model for Misuse Intrusion Detection", *Proceedings of the 17th National Computer Security Conference*, Oct. 1994, pp. 11-21.

[5] P. A. Porras and R. A. Kemmerer, "Penetration State Transition Analysis - A Rule-Based Intrusion Detection Approach", *Eight Annual Computer Security Applications Conference*, IEEE Computer Society Press, November 30-December4 1992, pp.220-229.

[6] K. Ilgun, R. Kemmerer and P. Porras, "State Transition Analysis: A Rule Based Intrusion Detection System", *IEEE Transactions on Software Engineering*, Mar. 1995, 21(3).

[7] F. Wang, F. Gong, F. S. Wu and H. Qi, "Design and Implementation of A New Intrusion Detection Approach: Property-Oriented Detection", *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, New York, June 4-5, 2001, pp. 91-99.

[8] D. E. Denning, "An Intrusion Detection Model", *IEEE Transactions on Software Engineering*, 13(2), 1987, pp. 222-232.

[9] G. E. Liepins and H. S. Vaccaro, "Anomaly Detection: Purpose and Framework", *Proceedings of the 12th National Computer Security Conference*, Oct. 1989, pp. 495-504.

[10] H. Javitz and A. Valdez, "The SRI IDES Statistical Anomaly Detector", *Proceedings of IEEE Symposium on Research in Security and Privacy*, May 1991, pp. 316-326.

[11] R. Jagannathan, T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neumann, A. Tamaru and A. Valdez, "System Design Document: Next Generation Intrusion Detection Expert System (NIDES). Technical Report A007/A008/A009/A011/A012/A014", SRI International, March 1993.

[12] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", *Proceedings of the 19th National Computer Security Conference*, Baltimore, MD, Oct. 1997, pp. 353-365.

[13] S. Stolfo, W. Fan, W. Lee, A. Prodromidis and P. Chan, "Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project", *Proc. DARPA Information Survivability Conference and Exposition*, IEEE Computer Press, 2000, pp. 130-144.

[14] Y. Yemini, A. Dailianas, D. Florissi and G. Huberman, "MarketNet: Market-based Protection of Information Systems", *Proceedings of ICE'98, First International Conference on information and Computation Economics*, Charleston, SC, Oct. 1998.

[15] T. Spyrou and J. Darzentas, "Intrusion Modeling: Approximating Computer User Intentions for Detection and Predictions of Intrusions", *Information Systems Security*, May 1996, pp. 319-335.

[16] T. Lane, "Hidden Markov Models for Human/Computer Interface Modeling", *Proceedings of the IJCAI'99 Workshop on Learning about Users*, 1999, pp. 35-44.

[17] T. Lane and C. E. Brodley, "Temporal Sequence Learning and Data Reduction For Anomaly Detection", *ACM Transactions on Information and System Security*, 2(3), 1999, pp. 295-331.

[18] S. Upadhyaya and K. Kwiat, "A Distributed Concurrent Intrusion Detection Scheme Based on Assertions", *SCS International Symposium on Performance Evaluation of Computer and Telecommunications Systems*, June 1999, pp. 369-376.

[19] J. Feldman, J. Giordano and J. Palmer, "Information Survivability at Rome Laboratory", *1997 IEEE Information Survivability Workshop*, 1997.

[20] S. Upadhyaya, R. Chinchani and K. Kwiat, "An Analytical Framework for Reasoning About Intrusions", *20th IEEE Symposium on Reliable and Distributed Systems*, New Orleans, LA, USA, Oct. 2001, pp. 99-108.

[21] K. Mantha, R. Chinchani, S. Upadhyaya and K. Kwiat, "Simulation of Intrusion Detection in Distributed Systems", *SCS Summer Simulation Conference*, July 2000.

[22] Claude Lecommandeur, *http://ctwm.dl.nu*.

[23] Robert Nation et. al., *http://www.fvwm.org*.

[24] The Open Group, *http://www.opengroup.org/cde*.

[25] G. C. Necula and P. Lee, "Proof-Carrying Code" *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Paris, Jan. 1997, pp. 106-119.

[26] R. Sekar, C. R. Ramkrishnan, I. V. Ramamkrishnan and Scott A. Smolka, "Model-Carrying Code (MCC): A New Paradigm for Mobile-Code Security", *New Security Paradigms Workshop (NSPW'01)*, Cloudcroft, New Mexico, Sept. 2001.

[27] C. Ko, M. Ruschitzka and K. Levitt, "Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach", *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 1997, pp. 175-188.

[28] A. Somayaji and S. Forrest, "Automated Response Using System-Call Delays", *Usenix Security Symposium*, 2000.

[29] K. Jain and R. Sekar, "User-Level Infrastructure for System Call Interception: A Platform for Intrusion Detection and Confinement", *Network and Distributed Systems Security Symposium (NDSS)*, 2000.

[30] T. Fraser, L. Badger and M. Feldman, "Hardening COTS Software with Generic Software Wrappers", *Symposium on Security and Privacy*, 1999.

[31] T. Mitchum, R. Lu and R. O'Brien, "Using Kernel Hypervisors to Secure Applications", *Annual Computer Security Conference*, Dec. 1997.