Application Intrusion Detection

Robert S. Sielken

Department of Computer Science
School of Engineering and Applied Science
University of Virginia
Charlottesville, VA 22903
rsielken@cs.virginia.edu

Abstract

Intrusion detection has traditionally been performed at the operating system (OS) level by comparing expected and observed system resource usage. OS intrusion detection systems (OS IDS) can only detect intruders, internal or external, who perform specific system actions in a specific sequence or those intruders whose behavior pattern statistically varies from a norm. Internal intruders are said to comprise at least fifty percent of intruders [ODS99], but OS intrusion detection systems are frequently not sufficient to catch such intruders since they neither significantly deviate from expected behavior, nor perform the specific intrusive actions because they are already legitimate users of the system.

We hypothesize that application specific intrusion detection systems can use the semantics of the application to detect more subtle, stealth-like attacks such as those carried out by internal intruders who possess legitimate access to the system and its data and act within their bounds of normal behavior, but who are actually abusing the system. To test this hypothesis, we developed two extensive case studies to explore what opportunities exist for detecting intrusions at the application level, how effectively an application intrusion detection system (AppIDS) can detect the intrusions, and the possibility of cooperation between an AppIDS and an OS IDS to detect intrusions. From the case studies, we were able to discern some similarities and differences between the OS IDS and AppIDS. In particular, an AppIDS can observe the monitored system with a higher resolution of observable entities than an OS IDS allowing tighter thresholds to be set for the AppIDS' relations that differentiate normal and anomalous behavior thereby improving the overall effectiveness of the IDS.

We also investigated the possibility of cooperation between an OS IDS and an AppIDS. From this exploration, we developed a high-level bi-directional communication interface in which one IDS could request information from the other IDS, which could respond accordingly. Finally, we explored a possible structure of an AppIDS to determine which components were generic enough to use for multiple AppIDS. Along with these generic components, we also explored possible tools to assist in the creation of an AppIDS.

Table of Contents

Acknowledgements

I would like to acknowledge the following people who have contributed to this work in one manner or another. Anita Jones, my advisor, was instrumental in guiding the direction of the research and rigorously analyzing every little detail of this work. Brownell Combs, fellow graduate student, served as a backboard off of which to bounce ideas, both realistic and ridiculous, about intrusion detection. And last but not least, Kaselehlia Sielken, my wife, who provided the emotional support to keep this work from driving me over the edge. To these people, I am extremely grateful.

1 Introduction

When a user of an information system takes an action that the user was not legally allowed to take, it is called *intrusion*. The intruder may come from outside, or the intruder may be an insider who exceeds his limited authority to take action. Whether or not the action is detrimental, it is of concern because it might be detrimental to the health of the system or to the service provided by the system.

As information systems have come to be more comprehensive and a higher value asset of organizations, *intrusion detection* subsystems have been incorporated as elements of operating systems, although not typically applications. Most intrusion detection systems (IDS) attempt to detect a suspected intrusion in the *monitored system* and then alert a system administrator. The technology for automated reaction is just beginning to be fashioned. Original intrusion detection systems assumed a single, stand-alone processor system, and detection consisted of post-facto processing of audit records. Today's systems consist of multiple nodes executing multiple operating systems that are linked together to form a single distributed system. Intrusions can involve multiple intruders. The presence of multiple entities only changes the complexity, but not the fundamental problems. However, that increase in complexity is substantial.

Intrusion *detection* involves determining that some entity, an *intruder*, has attempted to gain, or worse, has gained unauthorized access to the system. Casual observation shows that none of the automated detection approaches seek to identify an intruder *before* that intruder initiates interaction with the system. Of course, system administrators routinely take actions to *prevent* intrusion. These can include requiring passwords to be submitted before a user can gain any access to the system, fixing known vulnerabilities that an intruder might try to exploit in order to gain unauthorized access, blocking some or all network access, as well as restriction of physical access. Intrusion detection systems are used in addition to such preventative measures. Some system errors may appear to the intrusion detection system to be intrusions. But, the detection of these errors increases the overall survivability of the system, so unintentional detection will be considered desirable and not precluded.

To limit the scope of the problem of detecting intrusion, system designers make a set of assumptions. Total physical destruction of the system, which is the ultimate denial of service, is not considered. Intrusion detection systems are usually based on the premise that the operating system, as well as the intrusion detection software, continues to function for at least some period of time after an intrusion so that it can alert administrators and support subsequent remedial action. However, the intrusion detection system or the system itself may not operate at all after an intrusion thereby eliminating the opportunity for the intrusion detection system to mitigate possible damage.

Intruders are classified into two groups. *External intruders* do not have any authorized access to the system they attack. *Internal intruders* have some authority, but seek to gain additional ability to take action without legitimate authorization. Internal intruders may act either within or outside their bounds of authorization. Internal intruders are further subdivided into the following three categories. *Masqueraders* are external intruders who have succeeded in gaining access to the system. An employee without full access who attempts to gain additional unauthorized access, or

an employee with full access but who wishes to use some other legitimate user's identification and password would be considered a masquerader. *Legitimate* intruders have access to both the system and the data but misuse this access (misfeasors). *Clandestine* intruders have or have obtained supervisory (root) control of the system and as such can either operate below the level of auditing or can use the privileges to avoid being audited by stopping, modifying, or erasing the audit records if they so choose [Anderson80].

Intrusion detection systems have a few basic objectives that characterize what properties the IDS is attempting to provide. The following objectives from [Biesecker97] are indicative of the basic objectives of an IDS:

Confidentiality – ensuring that the data and system are not disclosed to unauthorized individuals, processes, or systems

Integrity – ensuring that the data is preserved in regard to its meaning, completeness, consistency, intended use, and correlation to its representation

Availability – ensuring that the data and system are accessible and usable to authorized individuals and/or processes

Accountability – ensuring that transactions are recorded so that events may be recreated and traced to users or processes

It is also assumed that intrusion detection is not a problem that can be solved once; continual vigilance is required. Complete physical isolation of a system from all possible, would-be external intruders is a simple and effective way of denying external intruders, but it may be unacceptable because physical isolation may render the system unable to perform its intended function. Some possible solution approaches cannot be used because they are in conflict with the service to be delivered.

In addition, potential internal intruders have legitimate access to the system for some purposes. So, it is assumed that at least insiders, and possibly outsiders, have some access and therefore have some tools with which to attempt to penetrate the system. It is typically assumed that the system, usually meaning the operating system, does have flaws that can be exploited. Today, software is too complicated to assume otherwise. New flaws may be introduced in each software upgrade. Patching the system could eliminate known vulnerabilities. However, some vulnerabilities are too expensive to fix, or their elimination would also prevent desired functionality.

Vulnerabilities are usually assumed to be independent. Even if a known vulnerability is removed, a system administrator may run intrusion detection software in order to detect *attempts* at penetration, even though they are guaranteed to fail. Most intrusion detection systems do not depend on whether specific vulnerabilities have been eliminated or not. This use of intrusion detection tools can identify a would-be intruder so that his/her other activities may be monitored. This type of monitoring is just one of the many methods available to detect other vulnerabilities.

Intrusion detection software mechanisms themselves are not inherently survivable; they require some protection themselves to prevent an intruder from manipulating the intrusion detection system to avoid detection. Most systems depend upon the assumption that the intrusion detection

system itself, including executables and data, cannot be tampered with. Some intrusion detection systems are based on state changes, and assume that initially the system being monitored is "clean", that is, no intrusion has occurred before the intrusion detection subsystem begins operation and completes initialization. Some of these problems can be solved by executing the intrusion detection system on a separate computer with its own CPU and memory so that it does not have to compete for regular system resources. However, the information about the observed system used to detect intrusions still resides on the observed system that could become corrupted. Therefore, the intrusion detection system must have some built-in survivability to handle the case of infrequent events or audit records and must be able to verify or assume that the incoming data arrives uncorrupted through trusted connections and/or encryption.

As with all computer programs, some fine-tuning is necessary. Ideally, an intrusion detection system would catch every intrusion (raise positive alarms) and ignore everything else (no false alarms indicating an intrusion when none exists). However, in the real world, this is not possible. So, the system must be configured with numerous parameters to try to maximize the number of positive alarms and minimize the number of false alarms. At this time, this continues to be more of an art than an engineering science. All the intrusion detection systems are thus susceptible to improper configuration and a false sense of security.

Intrusion detection has traditionally been performed at the operating system (OS) level by comparing expected and observed system resource usage. OS intrusion detection systems can only detect intruders, internal or external, who perform specific system actions in a specific sequence or those intruders whose behavior pattern statistically varies from a norm. Internal intruders are said to comprise at least fifty percent of intruders [ODS99], but OS intrusion detection systems are frequently insufficient to catch such intruders since they neither significantly deviate from expected behavior, nor perform the specific intrusive actions because they are already legitimate users of the system.

We hypothesize that application specific intrusion detection systems can use the semantics of the application to detect more subtle, stealth-like attacks such as those carried out by internal intruders who possess legitimate access to the system and its data and act within their bounds of normal behavior, but who are actually abusing the system. This research will explore the opportunities and limits of utilizing application semantics to detect internal intruders through general discussion and extensive examples. We will also investigate the potential for application intrusion detection systems (AppIDS) to cooperate with OS intrusion detection systems (OS IDS) to further increase the level of defense offered by the collective intrusion detection system.

2 State of Practice – OS IDS

OS IDS monitoring the resource usage of the operating system and the network represent the state of practice. They can only monitor the resource usage of the application and not the applications themselves. OS IDS typically obtain the values necessary to perform intrusion detection from the existing audit records of the system. These *audit records*, operating system generated collections of the events (normally stored in a file) that have happened in the system over a period of time, are then analyzed by the OS IDS to determine whether or not an intrusion has occurred. *Events* are the results of actions taken by users, processes, or devices that may be related to a potential intrusion. It is these events that are analyzed by the IDS either through the existing audit records or specially defined audit records that record other events that the existing audit records do not record. OS IDS can detect both external and some internal intruders. Identification is straightforward since audit records contain the identity of who caused an event. This identification is normally the user identification of the user who logged into the system and either issued the command or started the application process that caused the event on behalf of the user.

2.1 Threats that can be detected by an IDS

The following categories represent the general threats that the OS IDS can detect. While this may not be a completely comprehensive categorization, most intrusions will fall into one of these categories. The first six categories [Biesecker97] do not seem to cover the entire set of intrusions, so we have added two categories to the list to better cover the set of all intrusions.

Denial of Service – action or series of actions that prevent some part of a system from performing as intended

Disclosure – unauthorized acquisition of sensitive information

Manipulation – improper modification of system information whether being processed, stored, or transmitted

Masqueraders – attempt by an unauthorized user or process to gain access to a system by posing as an authorized entity

Replay – retransmission of valid messages under invalid circumstances to produce unauthorized effects

Repudiation – successful denial of an action

Physical Impossibilities – violation of an object residing in two places at the same time, moving from one place to another in less than optimal time, or repeating a specific action in less than some minimal time quantum

Device Malfunctions (health of the system) – partial or complete failure of a monitored system device

Any system with physical components will have some state(s) that the system should never attain. The second to last category, Physical Impossibilities, is meant to represent intrusions achieved by putting the system in such a state. Some errors may look like intrusions and vice-versa, so an IDS detecting either is beneficial since it will help the IDS increase the overall survivability of the system. For this reason, the last threat category, Device Malfunctions, was added.

2.2 Intrusion Detection Approaches

Currently there are two basic approaches to intrusion detection. The first approach is to define and characterize correct static form and/or acceptable dynamic behavior of the system and then to detect abnormal behavior by defining statistical relations. This is called *anomaly detection.* It relies on being able to define the desired form or behavior of the system and then to distinguish between that definition and undesirable form or anomalous behavior. While the boundary between acceptable and anomalous forms of stored code and data can frequently be precisely defined, the boundary between acceptable and anomalous behavior is much more difficult to define.

The second approach, called *misuse detection*, involves characterizing known ways to penetrate a system, usually described as a pattern, and then monitoring for the pattern by defining rule-based relations. The pattern may be a static bit string, for example, a specific virus bit string insertion. Alternatively, the pattern may describe a suspect set or sequence of events. Patterns are represented in a variety of forms as will be discussed later.

Intrusion detection systems have been built to explore both approaches: anomaly detection and misuse detection. In some cases, they are combined in a complementary way in a single intrusion detector. There is a consensus in the community that both approaches continue to have value.

The concept of intrusion detection appears to have been first used in the 1970's and early 1980's [Anderson80]. In what we will call the first generation of intrusion detection systems, operating system audit records were post-processed. Both anomaly detection and misuse detection approaches were invented early and persist until today. In the second generation, the processing became more statistically sophisticated, more behavior measures were monitored, and real-time alerts became possible. A seminal paper defining an early second-generation intrusion detection system implementation (IDES) appeared in 1987 [Denning87]. Variants of implementation offer advances, so that the quality of intrusion detection appears to be increasing. Host-based systems include Tripwire [Kim93, Kim94], Self-Nonself [Forrest94], NIDES [Anderson95a, Anderson95b, Javitz93, Lunt93], Pattern Matching [Hofmeyer97], MIDAS [Sebring88], and STAT [Porras92]. A third generation of intrusion detection systems used the basic concepts developed by the second generation to detect intrusions on networked systems consisting of many hosts. Network system IDS include DIDS [Snapp91], NADIR [Hochberg93], NSTAT [Kemmerer97], GrIDS [Staniford-Chen96], and EMERALD [Porras97].

2.2.1 Anomaly Detection

By definition, anomalies are not nominal or normal. The anomaly detector must be able to distinguish between anomalous and normal behavior. Anomaly detection can be divided into static and dynamic. A static anomaly detector is based on the assumption that there is a portion of the system being monitored that should remain constant. Usually, static detectors only address the software portion of a system and are based on the tacit assumption that the hardware need not be checked. System administration tools check physical component configurations and report change, so such tools will not be treated here. The static portion of a system is the code for the system and the constant portion of data upon which the correct functioning of the system depends.

Static portions of the system can be represented as a binary bit string or a set of such strings (such as files). If the static portion of the system ever deviates from its original form, an error has occurred or an intruder has altered the static portion of the system. Static anomaly detectors are said to check for data integrity. Tripwire [Kim93, Kim94] and Self-Nonself [Forrest94] are examples of IDS that perform static anomaly detection.

Dynamic anomaly detectors, such as NIDES [Anderson95a, Anderson95b, Javitz93, Lunt93] or Pattern Matching [Hofmeyer97], must have a definition of behavior to classify as normal or anomalous. Frequently, system designers employ the notion of event. Behavior is defined as a sequence (or partially ordered sequence) of distinct actions that cause events that are recorded in audit records. Since audit records of an operating system only record events of interest, then the only behavior that can be observed is that which results in an event in an audit record. Events may occur in a sequence. In some cases such as with distributed systems, partial ordering of events is sufficient. In still other cases, the order is not directly represented; only cumulative information, such as cumulative processor resource used during a time interval, is maintained. In this case, thresholds are defined to separate normal resource consumption from anomalous resource consumption.

Where it is uncertain whether behavior is anomalous or not (Figure 1), the system may rely on parameters so that during initialization, the parameter values can be set to reflect observed behavior. If uncertain behavior is not considered anomalous, then intrusive activity may not be detected. If uncertain behavior is considered anomalous, then system administrators may be alerted by false alarms, i.e. in cases where there is no intrusion.

Figure 1: Anomalous behavior must be distinguished from normal behavior.

The most common way to draw this boundary is with statistical distributions having a mean and standard deviation; statistical distributions could be represented by the parameter values and type of the distribution or a discrete representation of the distribution with a specified number of bins and a relative frequency of a value appearing in that bin. Once the distribution has been established, a boundary can be drawn using some number of standard deviations. If an observation lies at a point outside of the (parameterized) number of standard deviations, it is considered unacceptable and may indicate a possible intrusion. These distributions are used to build a profile for each entity in the system. A *profile* is a collection of statistical distributions and other relevant information used to characterize the normal, acceptable behavior of the entity. The profile may be static but most often modifies itself to maintain a better representation of the entity whose activities may change over time.

2.2.2 Misuse Detection

Misuse detection is concerned with catching intruders who are attempting to break into a system by exploiting some known vulnerability. Ideally, a system security administrator would be aware of all the known vulnerabilities and would eliminate them. However, as was mentioned earlier, an organization may decide that eliminating known vulnerabilities is cost prohibitive or unduly

constrains system functionality. In practice, many administrators do not remove vulnerabilities even when they might. Users who have a history of abuse of the system or who slowly modify their activity so that their profiles contain the abusive activity are nearly impossible to detect with anomaly detectors. So, misuse detection systems look for known intrusions irrespective of the user's normal behavior. Example misuse detection systems are NIDES [Anderson95a, Anderson95b, Javitz93, Lunt93], MIDAS [Sebring88], and STAT [Porras92].

The term *intrusion scenario* is used as a description of a known kind of intrusion; it is a sequence of events that would result in an intrusion without some outside preventive intervention. An intrusion detection system continually compares recent activity to the intrusion scenarios to make sure that someone or a combination of someones are not attempting to exploit known vulnerabilities. To do this, each intrusion scenario must first be described or modeled in some way. Typically, intrusion scenarios are quite specific.

The main difference between the misuse detection techniques is in how they describe or model the bad behavior that constitutes an intrusion. Initial misuse detection systems used rules to describe the events indicative of intrusive actions that a security administrator looked for within the system. Large numbers of rules can be difficult to interpret if the rules are not grouped by intrusion scenarios since making modifications to the rule set can be difficult if the affected rules are spread out across the rule set. To overcome these difficulties, alternative intrusion scenario representations were developed. These new rule organizational techniques include model-based rule organization and state transition diagrams. Better rule organization allowed the intrusion scenarios to be described in a more expressive and understandable way for the misuse detection system user. Since the system will need to be constantly maintained and updated to cope with new intrusion scenarios, ease of use is a major concern.

Misuse detection systems use the rules to look for events that possibly fit an intrusion scenario. The events may be monitored live by monitoring system calls or later using audit records. Although most systems use audit records, they would be fundamentally the same if they were collecting live system information.

Since the intrusion scenarios can be specified, misuse detection systems can monitor the events that mark the progress of an intrusion attempt, a partially ordered sequence of user actions designed to penetrate a system. During the sequence of steps, the intrusion detection system can anticipate the next step of the intrusion using the intrusion scenario. Given this information, the intrusion detection system can more deeply analyze the audit records to check for the next step or can determine that the intrusion attempt has proceeded far enough and that some intervening actions should be taken to mitigate possible damage. The model based and state transition diagram characterizations lend themselves to anticipation related activities.

2.2.3 Extensions - Networks

Intrusion detection on a network of machines is similar to that of intrusion detection on a single machine, but there are some differences. Intrusion scenarios are based on the events associated with the actions taken by entities, so extending the system to include entities on a network is not different besides the fact that the OS is willing to respond to the network users. By responding to

network users, other intrusion alternatives are created. The multiple users of a network system can work together as part of a *cooperative* intrusion. A *cooperative* intrusion is one where the intrusion scenario is implemented by more than one entity from possibly different locations. These entities may represent different humans or may be the same human using different user identifications. Casual experience shows that cooperative intrusions in a distributed system are becoming more frequent than single entity intrusions because cooperative intrusions provide more options for intrusive activity. The intruder(s) can use the multiple nodes of a system as an attempt to disguise their activities since all the activity on a single host may be characterized as intrusive but a few activities from the cooperative intrusion on a single host would most likely not be characterized as intrusive. Therefore, the intrusion detection system must be able to correlate events and relate users from all the nodes in the system that are potentially participating in a cooperative intrusion.

The audit data or system state information must still be collected and then analyzed on a host by host basis. The difference between the singular case and the network case is that the intrusion detection system must aggregate and correlate the information from multiple hosts and the network that does not already have its events recorded in audit records. To accomplish this task, the intrusion detection system can either apply a centralized approach in which all the information is collected in a single location and then analyzed, or it may use a decentralized (hierarchical) approach where local information is analyzed and only selected, vital information is shared between the intrusion detection system components.

Centralized network intrusion detection systems are characterized by distributed audit collection and centralized analysis. Most, if not all, of the audit data is collected on the individual systems and then reported to some centralized location where the intrusion detection analysis is performed. This approach works well for smaller network situations but is inadequate for larger networks due to the sheer volume of audit data that must be analyzed by the central analysis component.

The intrusion detection system must be able to distinguish between audit trails from different operating systems since some different intrusion scenarios may apply to each different operating system being run. An example of an intrusion that is operating system dependent is the *setuid* shell intrusion that is possible in SunOS but not in Solaris [Kemmerer97]. *Setuid* is used to change the user identification associated with a process and can be used in an attempt to conceal the intruder's identity. Accounting for OS dependencies is just one problem with performing centralized analysis on information collected from a collection of heterogeneous system components.

Systems indicative of the centralized approach include Distributed Intrusion Detection System (DIDS) [Snapp91], NADIR (Network Anomaly Detection and Intrusion Reporter) [Hochberg93], and NSTAT (Network State Transition Analysis Tool) [Kemmerer97] which was an extension of the original ideas of USTAT [Ilgun93].

Decentralized network intrusion detection systems are characterized by distributed audit data collection and distributed intrusion detection analysis. These systems can be modeled as hierarchies. Unlike the centralized network intrusion detection systems, these systems are better

able to scale to larger networks because the analysis component is distributed and less of the audit information must be shared amongst the different components.

For the decentralized approach, there must be some way of partitioning the entire system into different, smaller domains for the purpose of communication. A *domain* is some subset of the hierarchy consisting of a node that is responsible for collecting and analyzing all the data from all the other nodes in the domain; this analyzing node represents the domain to the nodes higher up in the hierarchy. Domains can be constructed by dividing the system up by geography or administrative control. Another basis to create the domains is by collecting systems with similar software platforms. Yet another way to partition the entire system is by the anticipated types of intrusions. For example, a database system may be partitioned into the parts that are susceptible to intrusions relating to the data itself (manipulation or disclosure) and that are most likely to encounter denial of service attacks.

The key to designing an effective decentralized system is deciding what information should and should not be propagated up the hierarchy and in what form that information should be aggregated. Some intrusions can only be detected at a lower level in the hierarchy while others can only be detected at a higher level. Determining what and how information is processed will greatly impact the effectiveness of the intrusion detection system.

Representative decentralized systems include GrIDS and EMERALD. The Graph-Based Intrusion Detection System (GrIDS) [Staniford-Chen96] succeeds the work done on DIDS. EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [Porras97] uses a three-layer hierarchical approach to large scale system intrusion detection as an extension of SRI's previous work in intrusion detection, IDES [Lunt89] and NIDES [Anderson95a, Anderson95b, Javitz93, Lunt93].

2.3 Generic Characteristics of IDS

After analyzing the approaches taken by IDS at the operating system and network levels, some generic characteristics of intrusion detection became apparent. To characterize OS ID and then compare it to Application Intrusion Detection (AppID), we first need to define some terminology that will allow us to discuss the characteristics of both more precisely. This terminology is similar to that used for prior software and hardware error detection research.

A *relation* is an expression of how two or more values are associated. An *observable entity* is any object, such as a user or system device, that has or produces a value in the monitored system that can be used in defining a relation. Examples of operating system level observable entities include the percentage of CPU time used in the last time quantum, the number of file blocks associated with a user, and the timestamp of the last modification to a file. There are two basic types of relations although some blending between the two is possible. *Statistical relations* compare the current value of an observable entity to a profile, a collection of statistical and other relevant information characterizing normal or anomalous behavior, and are most often used in anomaly detection. *Rule-based relations* relate the immediate or accumulated value to a predefined expected value. These relations require two or more values to be associated by some function. For example, one value may be required to be within a fixed range or some percentage of the

other. The rule-based relations are most often used in misuse detection where the values of the observable entities are compared to the intrusion scenario that represents a specific intrusion.

Thresholds can be set for the relations regardless of whether they are statistical or rule-based. *Thresholds* determine how the result of the relation will be interpreted; results outside of the threshold will be considered anomalous and results within the threshold will be considered normal. Thresholds are normally characterized by a certain number of standard deviations for statistical distributions or by a range, either fixed in size or as a percentage of the expected value, for rule-based analysis. A key question is how the discrepancy between the two values relates to the semantics of the system. For the operating system, below normal resource usage may be good in that the system is not being over worked or it may be bad in that the system should be doing work and it currently is not.

Ideally, all intrusions would be detected without false alarms. Recall that a false alarm is an indication of an intrusion in the absence of an intrusion. However, this ideal is typically unattainable as many of the events indicative of an intrusion, such as those events of an intrusion scenario, are also legitimate events under certain circumstances. Processing of false alarms causes unnecessary time and resources to be diverted from achieving the mission of the system, and system administrators to dismiss reports of intrusions. Therefore, the values of the thresholds must be set to detect as many intrusions as possible without an intolerable amount of false alarms; this is commonly referred to as *fine-tuning* the intrusion detection system and determines the *effectiveness* of the IDS. Tighter thresholds, permitting less discrepancy, allow for greater detection but at the risk of more false alarms. Looser thresholds produce fewer false alarms but potentially at the cost of diminished detection. The tightness of the thresholds is largely a function of the system semantics. A relation involving an entity with consistent, well-defined behavior will allow for a tighter threshold value than a relation involving an entity with less consistent, potentially erratic behavior.

The frequency with which a relation is evaluated can also impact the effectiveness of the intrusion detection system. It is possible for the intrusion detection system to evaluate all relations immediately after each event, but this may place an intolerable processing cost burden on the intrusion detection system that would cause the IDS to fall increasingly farther behind or require the monitored system to run at a slower pace to allow the intrusion detection system to process the events. Therefore, events are typically collected in audit records over a period of time. Audit records entries can be reduced by combining some events into a single entry for analysis. By reducing the total number of events that must be analyzed, the IDS has a higher probability of keeping pace with the monitored system. For example, a single, failed log-in attempt is most likely insignificant, but many failed log-in attempts over a relatively short period of time may indicate a possible intrusion. The period of time between audit record analysis may be determined using real time or logical time where the relations are evaluated after a certain number of events have occurred.

The effectiveness of the IDS also depends on how many values are correlated to each other. With many values to correlate to each other, it is easier to determine which of the values are truly anomalous. If one value is anomalous, then most of the relations involving that value should have results that lie outside of their thresholds. With multiple relations involving the observable

entities of the anomalous relation result, the IDS may be able to determine which value used in those relations is anomalous the anomalous one. Therefore, the more values or relation results that can be correlated to each other, the more effective the intrusion detection system will be at detecting intrusions or system device failures.

From the previous discussion of the centralized and decentralized network intrusion detection systems, it is apparent that hierarchies may exist in the intrusion detection system. Centralized network IDS have only two levels in the hierarchy. The lower level consists of all the hosts being monitored who all pass their audit records to a single intrusion detection host where the analysis is performed. This single, analysis host is the upper level in the hierarchy. Decentralized network IDS may have more than two levels in the hierarchy. At the lowest level are the hosts that run single host intrusion detection analysis to catch local intrusions. A subset of the local audit record information is then passed to the next higher level. This level analyzes the records for all the hosts reporting to this level. This level may then pass even more general information to the next higher level, and so on until the highest level is reached which does high-level intrusion detection over the whole system. These hierarchies normally correspond to the hierarchy present in the monitored system, but this is certainly not a requirement. The hierarchy may be based on the physical location of the hosts, the administrative domains of the hosts, or some other characteristic such as machine or operating system type.

OS IDS have been deployed for many years now and have had a chance to mature to some degree. We hypothesize that intrusion detection at the OS level will have characteristics of intrusion detection systems regardless of the level of the monitored system. Therefore, we will use these characteristics to provide the basis for analyzing the opportunity to perform intrusion detection at the application level.

3 Case Studies of Application Intrusion Detection

OS IDS have advanced since their inception. However, the rate of improvements to their effectiveness in detecting intrusions has probably decreased. Therefore, a significant change in the approach to intrusion detection is needed to further increase the effectiveness of the IDS. In this pursuit, we explore the possibilities of using the basic ID techniques from OS IDS and the additional knowledge from the application semantics to improve the effectiveness of intrusion detection. To guide this exploration, we have defined three questions for which to search for answers regarding Application Intrusion Detection Systems (AppIDS):

Opportunity – what types of intrusions can be detected by an AppIDS?
Effectiveness – how well can those intrusions be detected by an AppIDS?
Cooperation – how can an AppIDS cooperate with the OS IDS to be more effective than either alone?

Since the concept of intrusion detection at the application level is fairly new, there is a lack of established literature on the subject for use in answering these questions. Therefore, we have decided to use a case study approach that permits us to explore the possible in-depth benefits of performing intrusion detection at the application level. We chose two case studies. From these, we hope to glean some general concepts about application intrusion detection (AppID) and determine its viability. By developing the examples, we also hope to develop a possible method of reasoning about such systems more generally.

3.1 Electronic Toll Collection

Our first case study is that of an electronic toll collection (ETC) system. It is comprised of numerous devices interconnected to expedite the toll collection process on a highway system. Despite being specific to transportation, we felt that ETC is a particularly interesting example because of certain properties that it possesses. The system incorporates numerous devices distributed throughout the transportation infrastructure. These devices are somewhat complementary in that the values they record are related. Therefore, they may be used to crosscheck each other. The devices are interlinked in a hierarchical fashion. The ETC system differs from an OS in that it has these independent, but linked, devices from which it gathers data about the external behavior that it monitors. Since ETC is concerned with collecting tolls, it includes, by necessity, an accounting application. Given that the ETC system contains many devices distributed throughout the transportation infrastructure in a hierarchical fashion, we believe this system to be representative of many information systems which possess these properties because many systems contain similar sensors or data collection components connected in a similar fashion that could be used by an AppIDS.

3.1.1 Electronic Toll Collection System Description

The ETC system is comprised of numerous devices organized in a three level hierarchy. At the lowest level are the individual toll booth lanes and the equipment installed in each lane. A collection of adjacent toll lanes comprises a toll plaza, the middle level in the hierarchy. The toll management center is the central control headquarters that manages all of the system's toll plazas as well as any other devices from the highway system that are not directly related to the toll lanes

or toll plazas. The toll management center is a single node and constitutes the highest level in the hierarchy. Figure 2 shows a sample ETC hierarchy.

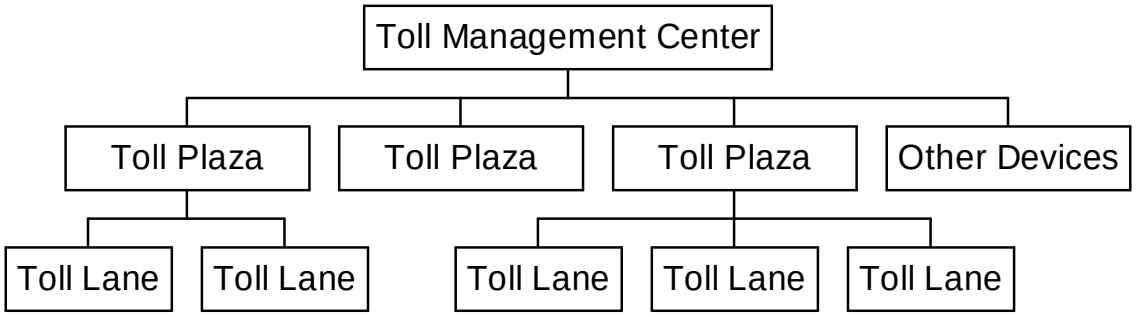

Figure 2: Representative ETC Hierarchy

The lowest level, the toll booth lane, contains most of the devices used in the collection of the electronic tolls. Each lane will have most, but possibly not all, of the following devices.

Tag Sensor – reads the tags to determine the ETC identification of the vehicle; located either overhead or underneath depending on the tag's location on the vehicle

Automated Coin Basket – a bin in which vehicle occupants toss change to pay the toll

Toll Booth Attendant – a person who collects tolls using a cash register (the attendant is frequently not used in an ETC system but could be)

Loop Sensor – device in the pavement that keeps a *vehicle count* (number of vehicles to pass over the sensor in a period of time) *occupancy* (percentage of time the sensor had a vehicle over it), and *speed* (average speed of the vehicles)

Axle Reader – device in the pavement that counts the number of axles of the vehicle

Weigh-In-Motion Scale – device in the pavement that weighs the vehicle while the vehicle is still in motion (speed may be obtained by the scale itself or through a loop sensor)

Traffic Signal – indicates to the driver whether or not the toll was collected successfully (normally a red-green traffic light)

Video Camera – mounted in a position so that is can take a snapshot of the rear of the vehicle (to record the license plate number) of all vehicles from which toll was not successfully collected

The toll booth lane must identify each vehicle as it passes through the lane. Therefore, vehicles have tags that interact with the toll booth lane's tag reader to identify the vehicle. The tags are the only devices that reside with the vehicles.

Tag – resides in the vehicle (such as on the dash) or on the vehicle (such as mounted underneath or on the windshield) for the purposes of identifying the vehicle to the tag sensor; tags come in two varieties – active and passive

Active tag – tag that emits a signal to the tag sensor that receives the signal

Passive tag – tag which is read by the tag sensor bouncing a signal off the tag and receiving the reflection[1]

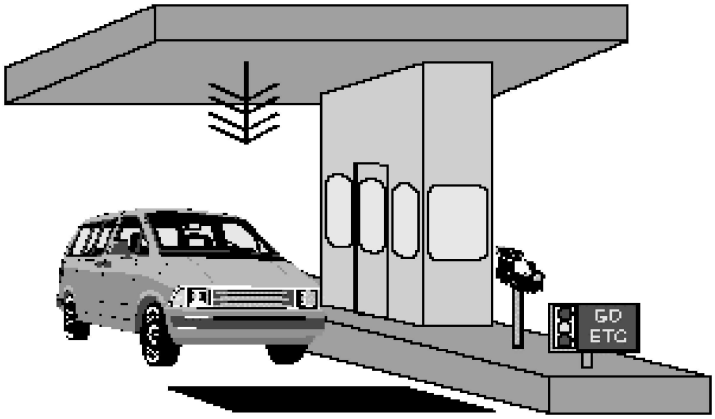

Figure 3 Representative ETC Toll Lane [ETTM98]

Besides the devices installed in the toll lanes or the vehicles, the transportation system may include other devices that could be useful in detecting intrusions. Loop sensors existing in other parts of the roadway may be useful as well. For the ETC case study, the only use of the loop sensors in other parts of the roadway the loop sensors' vehicle counts. These counts can be used in determining whether or not the system is functioning correctly since the loop sensors further down the highway should see roughly the same number of vehicles as the toll plaza (as long as there was not an entry or exit point between the toll plaza and the other highway loop sensors). Since these loop sensors are not associated with a toll lane, they will be part of the other devices part of the ETC hierarchy (Figure 2). All of the data collected from the devices is stored in a database; this database may be distributed throughout the hierarchy or at the Toll Management Center. The database will also contain the accounts of the ETC customers. Each account will be indexed by the tag identification and will contain the customer's personal information (name, address, etc.) as well as their account information such as current balance (most ETC systems are operated on a debit system) and a history of transactions. The data collection devices along with the computer systems that link these devices and maintain a database of system information and accounts comprise the ETC system. Figure 3 shows a sample toll lane with some ETC devices. The black patch in the road represents the axle reader and the weigh-in-motion scale. The antenna hanging from the roof over the vehicle is the tag sensor.

3.1.2 Application Specific Intrusions

To detect intrusions at the application level, one needs to first define relations that distinguish between normal and anomalous behavior. The relations are defined using the observable entities in the system. Recall that a relation is an expression of how two or more values are associated, and an observable entity is any object that has or produces a value in the monitored system that

1. There could be some argument as to whether or not this is actually a device. But the active tags certainly are devices, and ETC could not exist without the identification that the tags provide

can be used in defining a relation. Examples of observable entities in the context of the application include the value of a device reading, the value of a customer's account, the number of accesses to a user account over some period of time, and the frequency with which a system diagnostic process is activated. There are a large number of potential relations that could be defined using the observable entities in the system, but most of these relations will be irrelevant to intrusion detection. Therefore, the IDS needs to identify only the relevant relations. One possible way to determine which relations are relevant for ID is to consider potential *hazards*, real world harm caused by an intrusion(s), that could be caused by an intrusion of the system and find the relations that could possibly help detect those hazards. These relations may also help detect other hazards from other intrusions, either known or yet unknown. To find the potential hazards, we found it useful to use the same threat categories that were defined for OS ID to help derive the specific hazards for each application. These different threat categories help the derivation of these *specific hazards* by providing different perspectives from which to consider penetrating the system. Within each specific hazard there are different *methods* by which to execute the intrusion that causes the hazard. After the specific hazards and their corresponding methods have been derived from the threat categories, the AppIDS system designer can determine which of the possible relations could be used to detect each specific hazard. The relations that could detect a specific hazard are considered for inclusion in the AppIDS while the other relations will be eliminated from consideration since they do not increase the effectiveness of the AppIDS. Figure 4 gives a graphical representation of the basic steps in the process of identifying the relations from the generic threat categories. It should be noted that while this will be the usual progression of thought, some relations may be useful in defining alternative methods. Therefore, we do not eliminate this possibility even though it is not depicted in the figure.

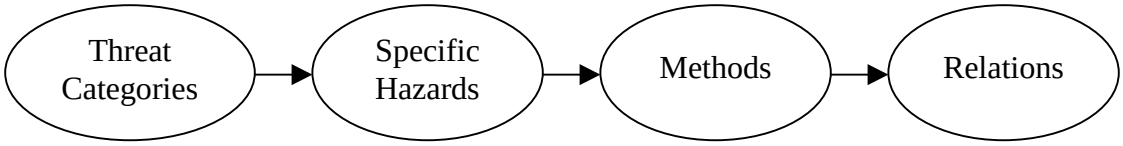

Figure 4 Relationships between the Threat Categories, Specific Hazards, Methods, and Relations

For this case study, we came up with the following five specific hazards caused by intrusions. These hazards and their methods are derived from the threat categories, so the threat category from which each hazard and method was derived is listed in parentheses after each method description. Since the relations must be able to be evaluated in order for the AppIDS to be effective in detecting hazards, the complete destruction of the ETC database is not considered since no relations can be evaluated without the observable entities that reside in the database being available. Database destruction and device malfunction are similar to the aforementioned ultimate denial of service, complete destruction of the monitored system, since the destruction of the database or a critical device is equivalent to the complete destruction of the AppIDS.

Hazard:	Annoyance
Methods:	Modify a random element of the database (Manipulation)
	Put many bogus packets on the network (Denial of Service)

Trojan device that performs its intended function but also performs some other, undesirable function (Disclosure)

Hazard: Steal electronic money
Methods: No tag and cover the license plate with a cover or dirt (Denial of Service)
Put a filter on the network that discards all of that tag's packets (Denial of Service)
Copy another tag for use (Masquerader)
Obtain account number and/or balance (Disclosure)
Make customer transparent by removing enough parts of the database so that the system will recognize the vehicle but not be able to determine which account to deduct from (Denial of Service)
Make the account read-only so that it will not deduct (Manipulation)
Have the tag deduct from some other account (Manipulation)
Change the deduction amount (Manipulation)
Transfer money from one account to another (Manipulation)
Add money to the ETC account without having any real money deducted from some other (presumably bank) account (Denial of Service)

Hazard: Steal vehicle
Methods: Change report of stolen vehicle (Manipulation)
No tag and cover the license plate with a cover or dirt (Denial of Service)
Put a filter on the network that discards all of that tag's packets (Denial of Service)
Copy another tag for use (Masquerader)

Hazard: Device Failure
Methods: Break a device either partially or totally (Denial of Service)

Hazard: Surveillance
Methods: Obtain personal information (Disclosure)
Obtain account number and/or balance (Disclosure)

3.1.3 Relation-Hazard Tables

After the specific hazards and methods have been derived from the threat categories, the relations that could detect the hazards caused by an intrusion need to be identified. The following five tables contain the specific hazards of the ETC system as well as the relations that could be used to detect them. The relations are numbered for identification of the relations between the different tables. Column two contains the name of the relation, and column three contains the description of the relation. Looking ahead to implementation issues, the description also contains whether or not it is more statistical (description concludes with "(stat)") or rule-like (description concludes with "(rule)") in nature. The execution location column (column four) indicates at what level(s) in the hierarchy of the system that the relation would be appropriate (TBL = Toll Booth Lane, TBP = Toll Booth Plaza, TMC = Traffic Management Center). The last column contains the specific hazard. The subdivisions of the last column (in the second row) indicate the different

methods by which the intrusion causing the hazard could be executed. An "X" in a box indicates that the relation from that row could potentially help detect the hazard caused by the intrusion carried out using the method of that column. All of the tables were originally combined into one large table by concatenating the last column of the last four tables to the last column of the first table, but space and readability constraints forced the tables to be separated here. Also, all of the relations are not applicable to all of the hazards, so the relations not applicable to a hazard (not having an "X" in the hazard's column) were omitted from the table for that hazard. For this reason, some tables will have discontinuous numbering in the relation number column (column one), but the ETC relations would be numbered consecutively from one to thirty-nine if the tables were recombined. Recall that there are three toll collection procedures: using the tag ("tag"), automated coin basket ("coin-toll"), and interaction with a toll booth attendant ("toll").

The following table (Table 1) shows the relations that can be used to detect an Annoyance hazard.

Table 1 Specific Relations for the Annoyance Hazard

Rel #	**Relation**	**Relation Description**	**Execution Location**	**Annoyance**	
				Bogus Packets	Trojan Device
25	Unreadable Tags	# of unreadable tags should be fairly evenly distributed between lanes and toll booths (stat)	TBP/TMC	X	
27	Personal information accesses	Accesses should follow a pattern per day/month/year (stat)	TMC	X	
28	Personal information changes	Personal information should generally not change; certainly not frequently (rule)	TMC	X	
29	Account information accesses	Accesses should follow a pattern per day/month/year (stat)	TMC	X	
30	Account information changes	Account information should generally not change; certainly not frequently (rule)	TMC	X	

From this table, we notice that there are five relations that could detect an Annoyance hazard that interjects bogus packets into the system but there are no relations that will detect a Trojan device in the system provided that it performs its intended function (as was assumed). It remains unclear whether or not a Trojan device will be detectable as long as it performs its intended function. If it does not perform that function, detection is fairly simple. Detecting the device that performs its intended function but also some other function may be impossible, it remains an open question, by an automated intrusion detection system. However, outside sources, such as customers calling to complain about somebody stealing their information and using it elsewhere, may be sufficient to detect such devices.

Stealing electronic money had so many methods that we divided the table into two parts, steal service and steal electronic money. Table 2 shows the relations that can be used to detect the hazard of stealing service from the ETC system.

Table 2 Specific Relations for the Steal Service Hazard

Rel #	Relation	Relation Description	Execution Location	Steal Service		
				No tag and cover plate	Copy another tag	Packet filter that discards all a tag's packets
1	Tag vs. Historical (Time)	Tag (Time of Day) should match Historical Time (of Day) (stat)	TBP/TMC		X	
2	Tag vs. Historical (Day)	Tag (Day of Week) should match Historical Time (Day of Week) (stat)	TBP/TMC		X	
3	Tag vs. Historical (Frequency)	Tag (Frequency (per day)) should match Historical Frequency (per day) (stat)	TBP/TMC		X	
4	Tag vs. Historical (Sites)	Tag (Sites) should match Historical sites (stat)	TMC		X	
5	Tag vs. Time	Tag should not be reread within *x* minutes any other toll both (rule)	TMC		X	
6	Tag vs. Parking	Tag (Identity) should not be listed as being in a parking facility (Parking) (rule)	TMC		X	
7	Tag vs. Report of Stolen Tag	Tag should not match that of a reported lost/ stolen vehicle (rule)	TMC		X	
9	Tag vs. Axles	Tag (Axles) should match Axles (rule)	TBL	X	X	X
10	Tag vs. Scale	Tag (Weight) should match Scale (rule)	TBL	X	X	X

Rel #	Relation	Relation Description	Execution Location	Steal Service		
				No tag and cover plate	Copy another tag	Packet filter that discards all a tag's packets
11	Tag + Toll + Coin Toll vs. Traffic Signal	# of tolls paid (tag/toll/coin-toll) equals number of signals given (green) (rule)	TBL			X
12	Tag + Toll + Coin Toll vs. Video	# of tolls paid (tag/toll/coin-toll) equals number of vehicles seen by camera (rule)	TBL	X		X
13	Tag + Toll + Coin Toll vs. Loops	# of tolls paid (tag/toll/coin-toll) equals number of vehicles seen by loops (rule)	TMC	X		X
15	Axles vs. Scale	# of Axles should match Scale reading (rule)	TBL		X	
16	Axles vs. Toll	Axles (cost) should match Toll collected (rule)	TBL	X	X	X
17	Axles vs. Coin-Toll	Axles (cost) should match Toll (coin) paid (rule)	TBL	X	X	X
18	Toll vs. Scale	Toll collected should match Scale based fare (rule)	TBL	X	X	X
19	Toll vs. Video	Toll collected should match Video vehicle determination (rule)	TBL	X	X	X
20	Coin-Toll vs. Scale	Toll (coin) paid should match Scale based fare (rule)	TBL	X	X	X
21	Coin-Toll vs. Video	Toll (coin) paid should match Video vehicle determination (rule)	TBL	X	X	X
22	Traffic Signal vs. Video	# of signals (green) equals # of vehicles seen by video camera (rule)	TBL	X		
23	Traffic Signal vs. Loops	# of signals (green) equals # of vehicles seen by loops (rule)	TMC	X		

Rel #	Relation	Relation Description	Execution Location	Steal Service		
				No tag and cover plate	Copy another tag	Packet filter that discards all a tag's packets
24	Video vs. Loops	# of vehicles seen by video camera equals # of vehicles seen by loops (rule)	TMC	X		
25	Unreadable Tags	# of unreadable tags should be fairly evenly distributed between lanes and toll booths (stat)	TBP/TMC	X		

From this table, we notice that there many different relations that can detect the theft of service hazard which is promising since charging customers for the use of the road is the reason there is an ETC system in the first place. The other interesting observation is that most of the relations are rule-like as compared to statistical in nature.

Table 3 shows the relations that can be used to detect the hazard of stealing electronic money from the ETC system. The effect is obtaining the service of using the road without ever having to really pay for the service. This hazard has the most methods, seven.

Table 3 Specific Relations for the Steal Electronic Money Hazard

Rel #	Relation	Relation Description	Exec Loc	Steal Electronic Money						
				Many small deposits into an account	Many small transfers from one account to another	Make customer transparent by removing parts of the database	Make account so that it won't deduct (read-only)	Have the tag deduct from some other account	Change the deduction amount (ex. $0.01)	Move money from one account to another
1	Tag vs. Historical (Time)	Tag (Time of Day) should match Historical Time (of Day) (stat)	TBP/ TMC			X				
2	Tag vs. Historical (Day)	Tag (Day of Week) should match Historical Time (Day of Week) (stat)	TBP/ TMC			X				
3	Tag vs. Historical (Frequency)	Tag (Frequency (per day)) should match Historical Frequency (per day) (stat)	TBP/ TMC			X				
4	Tag vs. Historical (Sites)	Tag (Sites) should match Historical sites (stat)	TMC			X				
5	Tag vs. Time	Tag should not be reread within *x* minutes any other toll both (rule)	TMC			X				
6	Tag vs. Parking	Tag (Identity) should not be listed as being in a parking facility (Parking) (rule)	TMC			X				
7	Tag vs. Report of Stolen Tag	Tag should not match that of a reported lost/stolen vehicle (rule)	TMC			X				

Rel #	Relation	Relation Description	Exec Loc	Steal Electronic Money						
				Many small deposits into an account	Many small transfers from one account to another	Make customer transparent by removing parts of the database	Make account so that it won't deduct (read-only)	Have the tag deduct from some other account	Change the deduction amount (ex. $0.01)	Move money from one account to another
8	Tag vs. Accounting	Tag (cost) should be equal to change in Accounting (rule)	TBP/ TMC			X	X			
9	Tag vs. Axles	Tag (Axles) should match Axles (rule)	TBL			X				
10	Tag vs. Scale	Tag (Weight) should match Scale (rule)	TBL			X				
11	Tag + Toll + Coin Toll vs. Traffic Signal	# of tolls paid (tag/toll/coin-toll) equals number of signals given (green) (rule)	TBL			X				
12	Tag + Toll + Coin Toll vs. Video	# of tolls paid (tag/toll/coin-toll) equals number of vehicles seen by camera (rule)	TBL			X				
13	Tag + Toll + Coin Toll vs. Loops	# of tolls paid (tag/toll/coin-toll) equals number of vehicles seen by loops (rule)	TMC			X				
14	Tag + Toll + Coin Toll vs. Accounting	Amount of tolls paid (tag/toll/coin-toll) equals amount of tolls collected (rule)	TBP			X	X			
15	Axles vs. Scale	# of Axles should match Scale reading (rule)	TBL			X				

Rel #	Relation	Relation Description	Exec Loc	Steal Electronic Money						
				Many small deposits into an account	Many small transfers from one account to another	Make customer transparent by removing parts of the database	Make account so that it won't deduct (read-only)	Have the tag deduct from some other account	Change the deduction amount (ex. $0.01)	Move money from one account to another
16	Axles vs. Toll	Axles (cost) should match Toll collected (rule)	TBL			X				
17	Axles vs. Coin-Toll	Axles (cost) should match Toll (coin) paid (rule)	TBL			X				
18	Toll vs. Scale	Toll collected should match Scale based fare (rule)	TBL			X				
19	Toll vs. Video	Toll collected should match Video vehicle determination (rule)	TBL			X				
20	Coin-Toll vs. Scale	Toll (coin) paid should match Scale based fare (rule)	TBL			X				
21	Coin-Toll vs. Video	Toll (coin) paid should match Video vehicle determination (rule)	TBL			X				
22	Traffic Signal vs. Video	# of signals (green) equals # of vehicles seen by video camera (rule)	TBL			X				
23	Traffic Signal vs. Loops	# of signals (green) equals # of vehicles seen by loops (rule)	TMC			X				
24	Video vs. Loops	# of vehicles seen by video camera equals # of vehicles seen by loops (rule)	TMC			X				

Rel #	Relation	Relation Description	Exec Loc	Steal Electronic Money						
				Many small deposits into an account	Many small transfers from one account to another	Make customer transparent by removing parts of the database	Make account so that it won't deduct (read-only)	Have the tag deduct from some other account	Change the deduction amount (ex. $0.01)	Move money from one account to another
25	Unreadable Tags	# of unreadable tags should be fairly evenly distributed between lanes and toll booths (stat)	TBP/ TMC			X				
26	Toll Charged	there is a minimum toll and a maximum toll (rule)	TBL			X			X	
27	Personal information accesses	accesses should follow a pattern per day/month/year (stat)	TMC		X	X				X
28	Personal information changes	personal information should generally not change; certainly not frequently (rule)	TMC			X				
29	Account information accesses	accesses should follow a pattern per day/month/year (stat)	TMC		X	X				X
30	Account information changes	account information should generally not change; certainly not frequently (rule)	TMC			X				

Rel #	Relation	Relation Description	Exec Loc	Steal Electronic Money						
				Many small deposits into an account	Many small transfers from one account to another	Make customer transparent by removing parts of the database	Make account so that it won't deduct (read-only)	Have the tag deduct from some other account	Change the deduction amount (ex. $0.01)	Move money from one account to another
31	Account credits	credits to accounts should be predictable in that most people will have similar deposits (most will deposit a similar amount on the same day each month) (stat)	TMC	X	X	X				X
32	Account credit amount	credits to accounts should be between the minimum and maximum allowable credit amount (rule)	TMC	X	X	X				X
33	Account credit frequency	credits should not be made more than once per *n* seconds (rule)	TMC	X	X	X				X
34	Account deductions	deductions from accounts should be predictable in that most people will have similar deductions (number and cost) per day/month/year (stat)	TMC		X	X		X		X
35	Account deduction amount	deductions should be between the minimum and maximum toll charge (rule)	TMC		X	X		X	X	X
36	Account deduction frequency	deductions should not be made more than once per *n* seconds (rule)	TMC		X	X		X		X

Rel #	Relation	Relation Description	Exec Loc	Steal Electronic Money						
				Many small deposits into an account	Many small transfers from one account to another	Make customer transparent by removing parts of the database	Make account so that it won't deduct (read-only)	Have the tag deduct from some other account	Change the deduction amount (ex. $0.01)	Move money from one account to another
37	# of vehicles (Day/Month/ Year) (vehicles - total, tagged, not tagged)	the number of different types of toll paying vehicles should be predictable (should be fairly constant taking into account days of the week and holidays) (stat)	TMC			X				
38	# of Transactions vs. # of Transactions	# of transactions from all TBL/TBP/ TMC should equal the # of transactions at the TBP/TMC (rule)	TBL/ TBP/ TMC			X				
39	Amount of Transactions vs. Amount of Transactions	sum of transactions from all TBL/ TBP/ TMC should equal the sum of transactions at the TBP/TMC (rule)	TBL/ TBP/ TMC			X	X			

This table yields many interesting observations. The first is that making the customer transparent by removing parts of the database so that the system cannot determine which account to debit could cause any of the relations to have anomalous results. This is consistent with the fact that any destruction, not just modification, of the database will cause the entire system to malfunction. Earlier, this was casually referred to as the ultimate denial of service. However, all of the other methods for creating a Steal Electronic Money hazard are detectable using some, but not all, of the relations.

The following table (Table 4) deals with an intruder trying to steal a vehicle. The thief is trying to steal the vehicle without getting noticed by the ETC system since that information could be useful to the law enforcement agencies in tracking down the thief.

Table 4 Specific Relations for the Steal Vehicle Hazard

Rel #	Relation	Relation Description	Execution Location	Steal Vehicle			
				Valid tag	No tag and cover plate	Packet filter that discards a tag's packets	Change report of stolen vehicle (delete from list)
1	Tag vs. Historical (Time)	Tag (Time of Day) should match Historical Time (of Day) (stat)	TBP/TMC	X			
2	Tag vs. Historical (Day)	Tag (Day of Week) should match Historical Time (Day of Week) (stat)	TBP/TMC	X			
3	Tag vs. Historical (Frequency)	Tag (Frequency (per day)) should match Historical Frequency (per day) (stat)	TBP/TMC	X			
4	Tag vs. Historical (Sites)	Tag (Sites) should match Historical sites (stat)	TMC	X			
7	Tag vs. Report of Stolen Tag	Tag should not match that of a reported lost/stolen vehicle (rule)	TMC	X			
9	Tag vs. Axles	Tag (Axles) should match Axles (rule)	TBL		X	X	
10	Tag vs. Scale	Tag (Weight) should match Scale (rule)	TBL		X	X	

Rel #	Relation	Relation Description	Execution Location	Steal Vehicle			
				Valid tag	No tag and cover plate	Packet filter that discards a tag's packets	Change report of stolen vehicle (delete from list)
11	Tag + Toll + Coin Toll vs. Traffic Signal	# of tolls paid (tag/toll/coin-toll) equals number of signals given (green) (rule)	TBL			X	
12	Tag + Toll + Coin Toll vs. Video	# of tolls paid (tag/toll/coin-toll) equals number of vehicles seen by camera (rule)	TBL		X	X	
13	Tag + Toll + Coin Toll vs. Loops	# of tolls paid (tag/toll/coin-toll) equals number of vehicles seen by loops (rule)	TMC		X	X	
16	Axles vs. Toll	Axles (cost) should match Toll collected (rule)	TBL		X	X	
17	Axles vs. Coin-Toll	Axles (cost) should match Toll (coin) paid (rule)	TBL		X	X	
18	Toll vs. Scale	Toll collected should match Scale based fare (rule)	TBL		X	X	
19	Toll vs. Video	Toll collected should match Video vehicle determination (rule)	TBL		X	X	
20	Coin-Toll vs. Scale	Toll (coin) paid should match Scale based fare (rule)	TBL		X	X	
21	Coin-Toll vs. Video	Toll (coin) paid should match Video vehicle determination (rule)	TBL		X	X	
22	Traffic Signal vs. Video	# of signals (green) equals # of vehicles seen by video camera (rule)	TBL		X		
23	Traffic Signal vs. Loops	# of signals (green) equals # of vehicles seen by loops (rule)	TMC		X		

Rel #	Relation	Relation Description	Execution Location	Steal Vehicle			
				Valid tag	No tag and cover plate	Packet filter that discards a tag's packets	Change report of stolen vehicle (delete from list)
24	Video vs. Loops	# of vehicles seen by video camera equals # of vehicles seen by loops (rule)	TMC		X		
25	Unreadable Tags	# of unreadable tags should be fairly evenly distributed between lanes and toll booths (stat)	TBP/TMC		X		

If the thief steals the car and uses the tag at the toll plazas that the vehicle normally uses, detection of the thief will be difficult. However, if the thief attempts to intrude the system to modify it in some way to avoid being detected, the number of relations that could detect the hazard caused by the intrusion is greatly increased.

The last table (Table 5) relates to the hazard of surveillance. Presumably, the intruder is attempting to gain some information (personal or account) or location (where was the last toll collected) of an ETC system user to be used for some other purposes, such as extortion.

Table 5 Specific Relations for the Surveillance Hazard

Rel #	Relation	Relation Description	Execution Location	Surveillance	
				Obtain personal information	Obtain account number and/or balance
27	Personal information accesses	accesses should follow a pattern per day/month/year (stat)	TMC	X	
29	Account information accesses	accesses should follow a pattern per day/month/year (stat)	TMC		X

Surveillance hazards are interesting because there are few relations that can be used to possibly detect them. If the intruder was clever with when and how often the information was obtained, the detection of the intruder is extremely difficult.

From this case study, we made some noteworthy observations. The main observation is that there are numerous relations that can be defined to detect a wide variety of hazards caused by intrusions executed by utilizing different methods. Some of the relations can detect anomalous behavior of the ETC customers, and some can detect system administrators who are abusing the system. There are more rule-based relations than statistical relations for the ETC application, but both types of relations appear to be useful in detecting intrusions at the application level. Only being able to evaluate relations at one level in the hierarchy would probably decrease the effectiveness of the IDS. But there are relations that can be evaluated at all the levels of the hierarchy and some relations that can be usefully evaluated at multiple levels. Not surprisingly, some hazards caused by intrusions can be detected by more relations than others can.

The ETC system is characterized by a large number of data collection devices distributed throughout the transportation infrastructure in a hierarchical fashion. Given the observable entities from the application and the additional semantics provided by the application, many relations can be defined and evaluated to detect hazards caused by intrusions and/or system errors at the application level that are not detectable at the operating system level.

3.2 Health Record Management

Given the close proximity of the University of Virginia Health Sciences Center, we chose to look at a subsystem of their large health care management system and develop a description of a hypothetical health record management (HRM) system based on that system to avoid breaching the security of their system by publishing details overly specific to their system. The subsystem we chose for out second case study was the HRM system that includes patient records, orders for drugs, tests, or procedures, and the schedules for filling these orders. Unlike the ETC example, this system is non-hierarchical, contains no devices beyond the controlling computer system

itself, does not contain a financial component. Only the medical staff (doctors, laboratory technicians, and nurses) and system administrators have legitimate access to the system. Similar to the ETC example, this system contains physical realities that can be used to crosscheck each other. This system is just representative of an information collecting and scheduling application that exists in many businesses.

3.2.1 Health Record Management System Description

The health record management system is comprised of three basic components. They are not hierarchical and operate on the same level as peer components. The basic components are:

Patient Records – contains all the information on the patients including name, address, phone number, insurance provider, drugs (medications), allergies, sex, blood type, and a chronological history of visits, test results, and surgical procedures

Orders – lists of all requests for drugs, tests, or procedures

Schedule – schedule for rooms for patient occupancy, laboratory tests, or surgical procedures (does not include scheduling of personnel such as doctors, laboratory technicians, or nurses)

The doctors, laboratory technicians, and nurses all interact through a user interface to access these components. Users can either search for information, enter new information such as recent test results, place an order, schedule a procedure, or generate a report of recent activity.

3.2.2 Application Specific Intrusions

As with the ETC case study, detecting intrusions at the application level requires that one first define relations that distinguish between normal and anomalous behavior. The relations are defined using the observable entities in the system. Examples of observable entities in the context of the HRM application include medication orders, the different pieces of the patient records, and the schedule for surgical procedures. Once again, we will follow a similar procedure as from the ETC case study to identify the relations relevant to detecting the real world hazards caused by intrusions. From the generic OS threat categories, specific hazards can be identified. From these specific hazards, methods by which to execute the intrusion to cause the hazard can be derived. Then, the relations that could possibly detect each specific hazard using the various methods can be identified.

For this case study, we derived the following four specific hazards. These hazards and their methods were derived from the threat categories, so the threat category from which each hazard and method were derived is listed in parentheses after each method description.

Hazard:	Annoyance
Methods:	Order Inapplicable Tests (Denial of Service)
	Order Inapplicable Procedures (Denial of Service)
	Order Inapplicable Drugs (Denial of Service)
	Schedule Conflicts (Denial of Service, Physical Impossibilities)

Hazard:	Steal Drugs
Methods:	Order Drugs for a Bogus Person (Manipulation, Replay)
Hazard:	Patient Harm
Methods:	Administer Wrong Drug (Manipulation, Replay)
	Administer Too Much of a Drug (Manipulation, Replay)
	Administer an Allergic Drug (Manipulation)
	Administer Improper Diet (Manipulation, Replay)
	Order Unnecessary Drugs (Manipulation)
	Perform an Unnecessary Procedure (Manipulation)
Hazard:	Surveillance
Methods:	Obtain personal information (Disclosure)
	Obtain account number and/or balance (Disclosure)

3.2.3 Health Record Management Relation-Hazard Tables

The following four tables contain the possible specific hazards caused by intrusions of the HRM system as well as the relations used to detect them. The structure of the tables is the same as that of the tables from the previous example, ETC, except for the execution location column has been removed since this system is non-hierarchical.

The following table (Table 6) shows the relations that can be used to detect an Annoyance hazard.

Table 6 Specific Relations for the Annoyance Hazard

Rel #	**Relation**	**Relation Description**	**Annoyance**			
			Order Inapplicable Tests	Order Inapplicable Procedures	Order Inapplicable Drugs	Schedule Conflicts
3	Drug vs. Sex	Certain drugs cannot be taken by one sex or the other (rule)			X	
9	Test vs. Sex	Certain tests are not applicable to one sex or the other (rule)	X			
10	Test vs. Test (different)	Some tests cannot be given too close to another test (rule)	X			

Rel #	Relation	Relation Description	Annoyance			
			Order Inapplicable Tests	Order Inapplicable Procedures	Order Inapplicable Drugs	Schedule Conflicts
11	Test vs. Test (same)	Some tests have a minimum time between the initial test and the next one (ex. Blood can only be drawn every ten weeks) (rule)	X			
13	Procedure vs. Sex	Certain procedures are not applicable to one sex or the other (rule)		X		
14	Procedure vs. Procedure (different)	Some procedures cannot be given too close to another test (rule)		X		
15	Procedure vs. Procedure (same)	Some procedures have a minimum time between the initial procedure and the next one (rule)		X		
17	Room vs. Room	A room cannot be scheduled for two different patients/ procedures/tests at the same time (rule)				X
22	Order vs. Time	Most orders should be done during daylight hours when the hospital is busier (stat)	X	X	X	
26	Patient Schedule	Cannot be scheduled for two tests/procedures/visits at the same time (rule)				X

Building this table yielded some interesting observations. Although the Annoyance hazard was also present in the ETC example, the methods used to execute an intrusion causing the hazard are different in the two case studies. Therefore, many hazards caused by intrusions may be similar from application to application, but since the methods are specific to the application, the methods will frequently be different. Like the ETC case study, the number of relations per method, about three, are similar. Also, some relations are useful in detecting hazards using different methods.

The following table (Table 7) deals with an intruder attempting to steal drugs. The thief is trying to steal the drugs by ordering drugs for a patient who does not really exist. When the drugs are delivered, the intruder or an accomplice would be there to pick up the drugs so that nobody else would notice. An intruder could modify an order for a drug by increasing the amount ordered and then taking the increase and leaving the original amount for the patient. However, this attack could be caught using these same relations, or the relations involved with the next specific hazard, Patient Harm.

Table 7 Specific Relations for the Steal Drugs Hazard

Relation Number	Relation	Relation Description	Steal Drugs
			Have Drugs Ordered for Bogus Person
8	Drug vs. Historical (dosage)	Drug dosage should be fairly similar to other prescriptions of the drug in either dosage amount or frequency (stat)	X
21	Order vs. Doctor	Doctors can only make certain orders based on their expertise (rule)	X

While this hazard has only one method, it does have two relations. The two relations are also balanced in that one is statistical and the other is more rule-like.

The following table (Table 8) deals with probably the most important hazard to the HRM system, Patient Harm. Any intrusion that can cause patients harm is of the utmost concern because negligent harm, whether caused by an intrusion or not, is detrimental to a health care provider, not to mention the legal liability engendered.

Table 8 Specific Relations for the Patient Harm Hazard

Rel #	Relation	Relation Description	Patient Harm					
			Admin. Wrong Drug	Admin. Too Much of Drug	Admin. an Allergic Drug	Admin. Improper Diet	Order Needless Drugs	Perform Needless Procedure
1	Drug vs. Drug	Certain drugs cannot be taken in conjunction with other drugs (rule)	X					
2	Drug vs. Allergy	Certain drugs cannot be taken when a person has certain allergies (rule)	X		X			
3	Drug vs. Sex	Certain drugs cannot be taken by one sex or the other (rule)	X					
4	Drug vs. Weight	Certain drugs prescriptions are based on the patient's weight (rule)	X	X				

Rel #	Relation	Relation Description	Patient Harm					
			Admin. Wrong Drug	Admin. Too Much of Drug	Admin. an Allergic Drug	Admin. Improper Diet	Order Needless Drugs	Perform Needless Procedure
5	Drug vs. Diet	Certain drugs cannot be taken while consuming certain foods (rule)	X			X		
6	Drug vs. Lethal Dosage	Drug dosage should not exceed the lethal dosage (rule)	X	X				
7	Drug vs. Time	Drugs have a minimum time between doses (such as 4 hours) (rule)	X	X				
8	Drug vs. Historical (dosage)	Drug dosage should be fairly similar to other prescriptions of the drug in either dosage amount or frequency (stat)	X	X				
12	Procedure vs. Diet	Some procedures may have a special dietary preparation requirement (rule)				X		
18	Language vs. Language	Anything outside of the restricted language is not allowed (rule)		X				
24	Patient Test Results vs. Test Results (Historical)	Test results should be related to previous test results for that patient (stat)	X	X			X	X
25	Test Results vs. Test Results (Historical)	Test results should be related to previous test results across all patients (stat)	X	X			X	X

Although this hazard is critical to detect, there are many (thirteen in all) relations that can be used to detect the hazard caused by an intrusion. Although a majority of the relations are rule-like, there are three that are statistical. It is also interesting to notice that relations reflect constraints of the real world such as constraints based on a patient's medical history, and general medical knowledge and practices.

The last hazard, Surveillance, is shown in the following table (Table 9).

Table 9 Specific Relations for the Surveillance Hazard

Rel #	Relation	Relation Description	Surveillance	
			Trojan Report Device	Obtain Personal Information
20	Reports vs. Location	Some reports should be sent regularly to the same site (stat)	X	
23	Doctor vs. Patient	Doctors should only access the information on their own patients (rule)		X
27	Patient Personal Information	Personal information such as name, address, social security number, eye color, blood type, etc. should not change (rule)		X
28	Patient Personal Information Records	There should be only one record for each patient (rule)		X
29	Records vs. Time	Records should be accessed in a pattern (stat)		X
30	Records vs. Time	A record not accessed in a long time (>12 months) may be allowed to be deleted (rule)		X
31	Staff Log-ins vs. Time	Staff log-ins should follow a pattern (stat)		X

Surveillance appears to be a hazard that will be applicable to most information systems that contain personal data of any sort. Since the trust of the customers, the vehicle operators in the ETC system or the patients in the HRM system, of the system is important to the operation of the system, they must be convinced that it is safe to trust the system with their personal or financial information. The last three relations (29-31) can be used to detect both users and maintainers of the system who are acting within their authorization but are actually abusing the system. With this table, all thirty-one of the relations for the HRM system have been presented.

While the HRM system is non-hierarchical, it has many physical constraints as well as observable entities from which to define relations that can detect hazards caused by intrusions as well as system errors that have effects similar to that of some intrusions. The existence of a restricted set of users of the system, no legitimate users outside of the health care professionals, does not appear to substantially limit the number of relations that can be used to detect the hazards since this case study had about as many relations as the ETC case study (both in the thirties).

In creating the tables for these two case studies, we have uncovered many interesting characteristics of AppIDS as well as a method to use to go about creating an AppIDS for other applications. The next two sections will further cover these two areas.

4 Application Intrusion Detection

After exploring these case studies, we were able to identify some of the similarities and differences between intrusion detection at the OS and application levels. The next subsection will discuss the differences. This will be followed by a discussion of how the AppIDS relies on the OS IDS for certain protections since AppIDS will be mostly concerned with anomaly detection and not misuse detection. The overriding goal is to improve the overall effectiveness of ID, so the last subsection will explore the possibilities of how the OS IDS and AppIDS might cooperate to accomplish this goal.

4.1 Differences between OS ID and AppID

Because OS ID and AppID are both concerned with detecting intruders, they both have the generic characteristics of IDS that were discussed in section 2.3. One similarity is that the AppIDS attempts to detect intrusions by evaluating relations to differentiate between anomalous and normal behavior. While AppIDS focus more on anomaly detection than misuse detection, the AppIDS can use both statistical and rule-based relations to detect the anomalies as was demonstrated by the relations in the tables of section 3. Also, the structure of an AppIDS may be similar to that of an OS IDS, centralized or decentralized (hierarchical). Given the reasonable amount of success in building the tables for the AppIDS using the OS threat categories defined in section 2.1, the same threat categories appear to be sufficient to guide the development of an AppIDS. However, there are certain characteristics that are unique to AppID. It is these differences that we wish to explore here.

AppIDS detect intruders in the context of the application. Therefore, the AppIDS only detects internal intruders after they either penetrated the OS to get access to the application or they were given some legitimate access to the application. The particular internal intruders of interest are the internal – legitimate [Anderson80] users who are basically *abusers* of the system. Another example of an abuser outside of the two case studies would be a credit card defrauder who uses the credit card correctly, scans the number and signs the slip, but is not the rightful owner of the card. Once an external intruder penetrates the operating system, he/she is no longer an external intruder but rather an internal intruder since he/she now has some inside access to the system. So, we make the assumption that all intruders are internal to simplify and focus the problem addressed by AppIDS. Because internal intruders already have some access to the monitored system, the application in this case, we will not consider intrusion scenarios that capture the original external intruders. Therefore, the AppIDS will be mostly concerned with anomaly detection. However, the AppIDS may utilize statistical as well as rule-based relations to detect the anomalies. Since the OS IDS must detect external intruders, AppIDS are, by definition, dependent on OS IDS; this and other dependencies will be further discussed in section 4.2.

Besides differences in the types of intruders detected, there is a difference in the two IDS in identifying the intruder once the intrusion has been detected. An OS is organized such that the process or the user that started the process or for whom the process was executed is associated with each event. However, applications may not be set up to perform this mapping between the event and the *event causing entity*, the observable entity that brought about the event. Recall that an observable entity is any object, such as a user or process, that has or produces a value in the monitored system that can be used in defining a relation. Event causing entities are only a proper

subset of the observable entities because they only produce (cause) a value, they do not necessarily have a value. An application may have its own identification and authentication procedure, but this identification may be established at the outset but then never used again or lost entirely. But since applications execute on top of operating systems, the OS IDS may be able to help the AppIDS determine the identity of the intruder. The issue of cooperation between the OS IDS and AppIDS for identification and other purposes will be discussed in section 4.3.

Because the results of relation evaluations detect the anomalous intrusions and relations are defined using the observable entities, more observable entities allow for more relations to be defined which may improve the effectiveness of the IDS. *Resolution* of the observable entities in the IDS is the relative size of the entities. An IDS that considers the monitored system as a few, large observable entities is said to have *lower resolution*, while an IDS that considers the monitored system as many, smaller observable entities is said to have *higher resolution*. The observable entities in the OS IDS are limited to values associated with resource usage because that is all the operating system manages, resources. In addition to observable entities based on resource usage, the AppIDS may have observable entities based on the correct functioning of the system characterized by the values that the application has stored in memory (the application's variables). For example, if data is stored in a file, then the OS IDS can only determine whether or not the file as a whole has changed. A file has an inherent *structure* and this structure can be frequently subdivided into *records* that in turn are divided into *fields*. The OS IDS cannot reason about changes to the structure or fields of the file since the OS IDS only views the file as a container whose contents cannot be deciphered except for changes in size. Therefore, the OS IDS has lower resolution than the AppIDS. The ability to define relations using resource usage and functional values and to subdivide larger entities into smaller entities gives the AppIDS the higher resolution. Because the two IDS may have many relations defined on the same basic observable entity, such as a file, although at different resolutions, there should be some correlation between events at the operating system and application levels. This correlation further provides a basis for the concept of cooperation between the two that will be discussed shortly (section 4.3).

The resolution of the AppIDS may be higher than the OS IDS, but this will be inconsequential if it does not aid in increasing the effectiveness of the overall IDS. Because it is easier to define relations that characterize the behavior of an observable entity whose actions or values vary less, the higher resolution of the AppIDS should allow for relations involving that observable entity to be defined with tighter thresholds than is possible in OS IDS. Relations could be defined based on a variety of dimensions such as the size of a file, number of transactions, and range of a value. In the case of a database file in the ETC example, the size of the file will be the same to both the OS IDS and the AppIDS. However, the number of transactions will be different because the OS IDS can only define a relation on the file as a whole, such as whether or not is was changed in the last period of time. The ETC AppIDS can define a relation on the different records of fields of the file. The ETC system may have thousands of transactions total per day, but an individual account may only be accessed four times per day. If the number of overall transactions on a standard workday varies by fifty transactions, this is probably insignificant. However, if the number of transactions on an individual account varies by more than ten transactions, it is probably indicative of an intrusion. Therefore, the AppIDS could define a relation with tighter thresholds on each account where a slight variation (ten) is significant while the OS IDS could only define a relation with looser thresholds on the overall number of account file changes where a variation of

ten would be insignificant and classified as normal behavior. The OS IDS does not have the resolution of the variables in the file, so the AppIDS is the only IDS that can define relations involving variables. Therefore, the increased resolution allows the AppIDS to define relations with tighter thresholds that would detect this intrusion that the OS IDS may have failed to detect because of its necessarily lower resolution and looser (less tight) thresholds.

Because the AppIDS relies on relations to detect intrusions, there must be some mechanism by which the AppIDS acquires the information necessary to evaluate the relations. OS IDS use the operating system audit records, operating system generated collections of the events (normally stored in a file) that have happened in the system over a period of time. Similarly, the AppIDS could build event records containing listings of all events and associated event causing entities of the application using whatever form of identification was available as was discussed previously. These event records could be generated in two ways. The first involves periodically, such as hourly or daily, scanning the system values and recording all of those values that have changed as entries in an event record. An example from the HRM case study would be scanning the list of drugs for each patient; any change in the drugs for a patient would be noted and then the appropriate relations (those involving "Drug") would be evaluated. The second option involves inserting *code triggers*; code triggers are small segments of code inserted into the application itself such that when certain events occur in the application, indicated by the code trigger being executed, it causes an event record entry to be generated or a relation to be evaluated. In the HRM case study, a code trigger may be inserted in the application such that any time a pulse rate is entered for a patient an event record entry is generated so that relations involving this new reading and prior readings may be evaluated. Given these event records, the AppIDS could perform the relation evaluations with a procedure similar to that which the OS IDS employs to evaluate the relations. Since the AppIDS operates with a higher resolution, the large number of observable entities may cause event records to become rather large in a short period of time. The large number of observable entities will also cause many relations to be evaluated many times using each of the observable entities. For example, a relation regarding the maximum account deduction from an ETC account would have to be run on each account. Therefore, the higher resolution of the AppIDS may hamper the performance of the AppIDS by creating more event record entries and forcing more relation evaluations than the AppIDS can handle in a timely manner. To counteract this potential problem, there are some optimizations that could reduce the number of event record entries or relation evaluations with a modest reduction in the effectiveness of the IDS. Once such optimization is not recording each event immediately following its occurrence but rather checking for any changes on a twice daily, daily, or weekly basis. Hence, many daily changes may be encapsulated into one event record entry for the entire week thereby significantly reducing the amount of event record information that must be processed and the number of times a relation must be evaluated.

4.2 Dependencies between OS IDS and AppIDS

In this section, we explore some of the dependencies between the two kinds of IDS. In particular, we explore the dependencies of the two on each other. Many of these dependencies, as mentioned in section 1, regard the protection of the IDS themselves. OS IDS have been deployed for years without AppIDS, so OS IDS are certainly not dependent on AppIDS. Since all applications run on top of operating systems, the AppIDS will rely on the OS IDS to provide some basic security

services such as protection of the application's code, data, and communications. The OS IDS must keep the application files from being changed by any means other than through the application itself thereby preventing the intruder from bypassing the AppIDS altogether by changing the files at the OS level. While the AppIDS may be able to detect such changes, the record of the event that changed the data may never be generated at the application level making it extremely difficult for the AppIDS to detect the change. Another dependency of the AppIDS is on the OS IDS to detect the external intruders as was mentioned in section 4.1. An AppIDS may be able to detect external intruder but only after the intruder gains access to the system at which point the intruder is reclassified as an internal intruder.

4.3 Cooperation between OS and App IDS

Because the intruder may cause audit record entries as well as event record entries to be generated that may cause relations to be evaluated, it is possible that one or both of the IDS could detect the intruder through relation evaluations. Therefore, the next logical avenue of exploration is how the two could cooperate to improve the overall of effectiveness of ID. For example, an AppIDS may detect a manipulation intrusion but cannot determine the identification of the intruder. However, the OS IDS may be able to identify the intruder. The reverse situation may also arise. An OS IDS may detect that an application has had a sudden increase in the number of files created. However, the OS IDS cannot determine whether this should or should not be happening, but the AppIDS may be able to determine as such since it has the additional information of the application semantics.

Correlation between audit and event record entries may be possible because both an audit record entry and an application event record entry may be generated by the same process or user action. In an AppIDS where relations are evaluated using the record entries, there may also be a correlation between the relation results of relations evaluated by the two IDS. This correlation of information may allow the two to cooperate by sharing information with each other. The communication between the two could be either uni-directional where one IDS periodically sends information to the other or bi-directional where one IDS can ask the other IDS for information in a client and service provider fashion.

If the two IDS are to cooperate, there must be some mechanism for the two to communicate with one another. For communication between the OS IDS and the AppIDS, there must be an interface between the two. Here, we explore one possible communication interface that permits one IDS to ask the other IDS for service in a client and service provider fashion. Current interfaces between OS IDS pass audit record information from one analysis engine to another analysis engine. However, this information is passed simply as facts in an audit record. There are no explicit requests for service or information to be returned although a somewhat related audit record entry may be generated that could end up back in the audit record information of the initial IDS [Porras97]. Since cooperative IDS will need to be able to not only pass information but also request services that may or may not have explicit returned information, they will need to have bi-directional information flows. The initiating IDS, the one that makes the request, will need to create a *bundle* of application information to send to the servicing IDS, the one that receives the request. A *bundle* is a collection of information to either request information from another IDS or to respond to another IDS's request. The *request bundle* may include the information request,

event descriptions, event times, and possible identifications that the other IDS could use to determine the appropriate response. The other IDS will then determine the answer to the request and form a *response bundle* to send back to the requesting IDS. For example, the AppIDS may determine that an event caused a relation result to fall outside of its threshold but the AppIDS does not know the identity of the event causing entity. Therefore, it creates a request bundle of the event and the event time to send to the OS IDS. The OS IDS takes this information, analyzes its audit records to determine the set of possible event causing entities, and builds the response bundle that is sent back to the AppIDS. Using this identification, the AppIDS may determine that it needs the OS IDS to assist in avoiding further damage since the AppIDS relies on the OS for some of its basic protections as discussed in section 4.2. The AppIDS would send a request bundle to the OS IDS asking for the OS IDS to increase protection of part or all of the application using the means at the OS IDS' disposal such as modifying access controls or completely removing the intrusive entity from the system by logging out the entity. This protection service would then be performed by the OS IDS. As this example illustrates, the interface between two IDS needs to support both information flows as well as service requests and responses, but there may be occasions were requests only flow in one direction such as a request for increased protection.

Although the communication between the two with bi-directional information flows seems simple enough, there are other complications. The application and the OS operate at different levels, so they must be able to communicate in terms that the other can understand. The OS deals only with system resources and their usage; it does not understand application specific concepts such as modification of the test results of a medical record. Therefore, the AppIDS will need to express its request bundles in terms that the OS can understand. The OS response bundle will be in terms of resource usage since that is all the OS understands, so the AppIDS will have to be capable of interpreting the response bundle of OS terms. Likewise, if one were to construct an OS IDS that could request information from an AppIDS, the OS IDS would need to be able to express its requests and interpret the responses in terms that the AppIDS could understand. Without being able to express requests and interpret response from the other domain, communications between OS IDS and AppIDS will be fruitless since the response receiving IDS will only receive what appears to be incomprehensible garbage. The lowest common denominator is resource usage, so this appears to be the simplest starting point for developing the communication interface.

5 Construction of an AppIDS

After deciding that intrusion detection at the application level could be theoretically possible, the next question is how practical it is to implement the AppID concepts. To explore the answer to this question, we will discuss the generic structure of an AppIDS as well as some possible tools that could be used to tailor each AppIDS to its application. These concepts are only developed in theory here; no actual implementation is implied nor are we aware of any actual implementations of these generic components or tools. However, construction of these items is not so far fetched as to be impractical.

An AppIDS detects intrusion by collecting information about the monitored application system, analyzing this information by evaluating relations, and taking some action if a relation result is judged to be anomalous. Because this applies to all AppIDS, a portion of an AppIDS can be generic and reused for the construction of multiple AppIDS. Specifically, the generic portion of the system has three components that relate to the three basic steps involved in detecting an intrusion. We now develop a notion of an AppIDS system with the objective of maximizing the amount of reusable intrusion detection code, thereby enabling the use of automated tools in the construction of the unique AppIDS.

The first generic component is the *event record manager.* It is responsible for the timely creation and collection of event records as well as maintaining the historical information necessary for the statistical relations. The event record manager will control how often entries in the event records are added to the event records: after each event occurrence, after a certain number of events, or after some period of time. Maintaining the historical information involves adjusting the statistical profiles on which the statistical relations rely to incorporate the new values from the most recent event records; this is necessary to keep the statistical profiles up to date. If the AppIDS is hierarchical, this tool will also need to manage the hierarchy to maintain and/or modify the event record information that gets shared between the levels of the hierarchy.

Once the relations have been specified (a tool for relation specification will be discussed shortly) and the event records collected, the evaluation of the relations is fairly generic between applications. The *relation evaluator* needs to know how to evaluate the two different types of relations, rule-based and statistical. As with the event record manager, the relation evaluator can be done on a real time schedule such as hourly or on a logical time schedule such as after each ten thousand events.

If the relation evaluator evaluates a relation and determines the result to be anomalous, the AppIDS will need a procedure to handle the anomalous relation result. The choices of what an AppIDS could do include alerting someone such as a system administrator or adapting the behavior of the application such as reducing the functionality of the application to avoid further damage. The generic code would have both options available. The generic component that handles the alarms is called the *anomaly alarm handler.*

These generic components will work for each AppIDS, but there will need to be a set of tools that assist in tailoring the AppIDS to the unique application to be monitored. These three tools will be

used in constructing and updating the AppIDS but will not be involved in the detection of intrusions because detection is done by the three generic components.

The first step is to determine which of the possible relations are useful to an AppIDS in detecting an intrusion. By building a table similar to the ones that were generated for the two case studies, the AppIDS designer identifies these relations. The basic steps involved in this process are (see also Table 10):

Use the generic threat categories to identify specific hazards caused by intrusions to the application (row one)
Determine methods to cause these specific hazards (row two)
List the possibly applicable relations (column two)
Explain the relation including its type (statistical or rule-based) (column three)
Determine at what levels the relation could be applicable if the system is hierarchical (column four)
Put X's where a relation may detect one of the specific intrusions

Table 10 Generic Table used in Developing an AppIDS

Relation Number	**Relation**	**Relation Description**	**Execution Location**	**Specific Hazard #1**		**Specific Hazard #2**	
				Method #1A	Method #1B	Method #2A	Method #2B
1	Relation Name #1	Relation Description #1	Loc_1/Loc_2		X	X	
2	Relation Name #2	Relation Description #2	Loc_2				X

Once the relevant relations have been identified, they need to be expressed in some format such that the relations can be implemented in code with minimal effort on the part of the AppIDS developer. Relations are of two varieties, statistical and rule-based, so the types of relations that need to be expressed are limited. Along with these relations must be the parameters that the relation may use to determine whether or not a relation result is classified as normal or anomalous. These parameters may be an exact value, a range of values, or a statistical distribution with its associated parameters (distribution, mean, variance, initialization values, and data aging function). The user specifies these relations using a tool called the *relation specifier.* Similar tools have been created for OS IDS such as MIDAS [Sebring88]. MIDAS uses the Production-Based Expert System Toolset (P-BEST) that is a forward-chaining, LISP based development environment for developing, compiling, and debugging rule-based relations.

The *relation-code connector* is a tool that takes the specified relations and identifies the relevant portions of the application code necessary to evaluate the relations. With the relations specified in a given format and the symbol table from the compiler of the application, the observable entities from the application to which the relations refer should be established. Once these ties have been

established, the AppIDS will be able to obtain the values of the observable entities necessary to evaluate the relations. As was mentioned in section 4.1, these values may be obtained by observing some value stored in a file and generating an event record entry or by inserting a code trigger into the application that causes an event record entry to be created or a relation to be evaluated.

The last tool needed to help build an AppIDS is the *event record specifier*. It assists the IDS builder in specifying the structure of the event records. An event record may contain a variety of fields, but the most common fields should be for the observable entity that was modified, the identity (if available) of who modified it, a timestamp, the old value of the observable entity, and the new value. Depending on the application, there may need to be more or less fields in the event record. This tool will also need to establish the timings for the collections of the event record entries such as after each event or after a certain period of time.

We have defined three tools that will help automate the development of code and configuration of the system so that the generic components may be reused in many AppIDS. This reuse is important because all AppIDS evaluate relations to detect intrusions. Constructing an AppIDS using the three generic components and the three tools, as opposed to constructing the AppIDS from the ground up for each application, may save significant amounts of development time. A relation evaluator for the ETC and HRM systems is basically the same once the information needed for evaluating the relations is collected. Figure 5 shows how the application specific tools that will help tailor the generic components to each application.

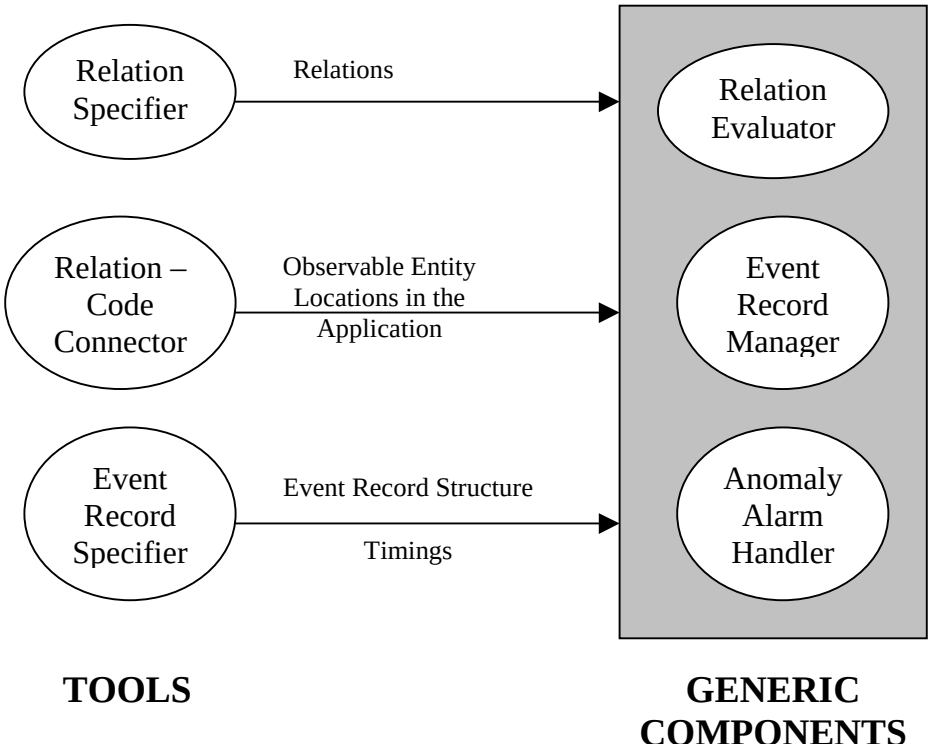

Figure 5 AppIDS Tools and Generic Components

6 Conclusions and Future Work

By exploring the possibilities of intrusion detection at the application level, we hoped to explore the opportunities and limits of utilizing application semantics to perform intrusion detection. From reviewing the state of practice for OS IDS and two extensive case studies, we hoped to determine what opportunity existed for intrusion detection to be done at the level of the application. We determined that application semantics can be used to detect internal intruders of an application using similar techniques to those employed by OD IDS. From the case studies, we found that the AppIDS will be mostly concerned with anomaly detection but use both forms of relations, statistical and rule-based, to do so. Although many of the categories of detectable threats were the same for the AppIDS as for the OS IDS, the higher resolution of observable entities with which the AppIDS operates allows it to detect intrusions that the OS IDS may not be able to detect because the OS IDS cannot utilize the semantics of the application. Hence, the AppIDS can be more effective at detecting the more subtle intrusions that the OS IDS could not detect because it operates with a lower resolution. Since the AppIDS relies on the OS or OS IDS for some basic protection and only the OS IDS can detect the external intruders, there is clearly a need for both an OS IDS and an AppIDS. If both IDS are present, there is potential for the two to cooperate in detecting some intrusions. We proposed a potential communication interface involving request and response bundles in which one IDS can request information from the other IDS.

Little research has been done into performing intrusion detection at the application level, so there is a general lack of literature on the subject. We have shown that the approach appears to be useful, but there still needs to be a considerable amount of work done in the area. In particular, many more application case studies would be useful to further verify or disprove some of the conclusions of this work. Our case studies had many characteristics common to many information systems such as numerous information sources distributed throughout a system in a hierarchical (ETC) or non-hierarchical (HRM) fashion in which access to the system is highly (HRM) or minimally (ETC) restricted. Additional case studies should cover information systems with other characteristics such as those involving real-time scheduling. The construction of an actual AppIDS would also be beneficial in furthering these conclusions. An actual AppIDS could help determine how generic the tools for constructing AppIDS could be, what resolution is feasible to avoid data overrun, and potentially qualify or quantify the gain in effectiveness that an AppIDS would have over an OS IDS. The other area in need of further research is that of communication between IDS. We developed a high-level communication interface with bi-directional information flows between the two IDS, but further research is necessary to determine the feasibility of one IDS communicating with another IDS in terms that either it or the other IDS understands. This could also be extended to include communication between two AppIDS instead of just between an AppIDS and an OS IDS.

7 References

[Anderson80] Anderson, J.P. “Computer Security Threat Monitoring and Surveillance.” Technical Report, James P. Anderson Co., Fort Washington, Pennsylvania, April 1980.

[Anderson95a] Anderson, D., T. Lunt, H. Javitz, A. Tamaru and A. Valdes. “Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES).” *SRI International Computer Science Laboratory Technical Report SRI-CSL-95-06*, May 1995.

[Anderson95b] Anderson, D., T. Frivold and A. Valdes. “Next-generation Intrusion Detection Expert System (NIDES): A Summary.” SRI International Computer Science Laboratory Technical Report SRI-CSL-95-07, May 1995.

[Biesecker97] Biesecker, Keith, Elizabeth Foreman, Kevin Jones and Barbara Staples. “Intelligent Transportation Systems (ITS) Information Security Analysis.” United States Department of Transportation Technical Report FHWA-JPO-98-009, November 1997.

[Denning87] Denning, D. “An Intrusion Detection Model.” *IEEE Transactions on Software Engineering*, 13.2 (1987) 222.

[ETTM98] ETTM On The Web. “Electronic Toll Collection (ETC).” September 1998, http://www.ettm.com/etc.html.

[Forrest94] Forrest, S., L. Allen, A.S. Perelson, and R. Cherukuri. “Self-Nonself Discrimination in a Computer.” *Proceedings of the 1994 IEEE Symposium on Researh in Security and Privacy*, 1994.

[Hochberg93] Hochberg, J., K. Jackson, C.Stallings, J.F. McClary, D. DuBois, and J. Ford. “NADIR: An Automated System for Detecting Network Intrusion and Misuse.” *Computers and Security*, 12.3 (1993) 235-248, http://nadir.lanl.gov/libLA-UR-93-137.html.

[Hofmeyer97] Hofmeyer, S.A., S. Forrest, and A. Somayaji. “Intrusion Detection using Sequences of System Calls.” Revised: December 17, 1997. http://www.cs.unm.edu/~steveah/publications/ids.ps.gz.

[Ilgun93] Ilgun, K. “USTAT: A Real-Time Intrusion Detection System for UNIX.” *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1993.

[Javitz93] Javitz, H.S. and A. Valdes. “The NIDES Statistical Component: Description and Justification.” ftp://ftp.csl.sri.com/pub/nides/reports/statreport.ps.gz, March 1993.

[Kemmerer97] Kemmerer, R.A. “NSTAT: A Model-based Real-time Network Intrusion Detection System.” *University of California-Santa Barbara Technical Report TRCS97-18*, November 1997.

[Kim93] Kim, G.H. and E.H. Spafford. “A Design and Implementation of Tripwire: A File System Integrity Checker.” *Purdue Technical Report CSD-TR-93-071*, November 1993.

[Kim94] Kim, G.H. and E.H. Spafford. “Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection.” *Purdue Technical Report CSD-TR-94-012*, February 1994.

[Lunt89] Lunt, T., R. Jaganathan, R. Lee, A. Whitehurst and S. Listgarten. “Knowledge-Based Intrusion Detection.” *Proceedings of the 1989 AI Systems in Government Conference*, March 1989.

[Lunt93] Lunt, T.F. “Detecting Intruders in Computer Systems.” *1993 Conference on Auditing and Computer Technology*, 1993.

[ODS99] ODS Networks, Inc. “Extreme Access . . . Infinite Possibilities.” *ODS Networks White Paper*, March 1999, http://www.ods.com/white/whi_0004.shtml.

[Porras92] Porras, P.A. and R.A. Kemmerer. “Penetration State Transition Analysis: A Rule-Based Intrusion Detection Approach.” *Proceedings of the Eighth Annual Computer Security Applications Conference*, December 1992.

[Porras97] Porras, P.A. and P.G. Neumann. “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances.” *19th National Information System Security Conference (NISSC)*, 1997, http://www.csl.sri.com/emerald/emerald-niss97.html.

[Sebring88] Sebring, M.M., E. Shellhouse, M. Hanna and R. Whitehurst. “Expert Systems in Intrusion Detection: A Case Study.” *Proceedings of the 11th National Computer Security Conference*, October 1988.

[Snapp91] Snapp, S.R., J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, C. Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, D.M. Teal and D. Mansur. “DIDS (Distributed Intrusion Detection System) – Motivation, Architecture, and An Early Prototype.” *Proceedings of the 14th National Computer Security Conference*, October 1991.

[Staniford-Chen96] Staniford-Chen, S., S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. “GrIDS – A Graph Based Intrusion Detection System for Large Networks.” *20th National Information System Security Conference (NISSC)*, October 1996, http://olympus.cs.ucdavis.edu/arpa/grids/nissc96.ps.