INTRUSION DETECTION FOR DISTRIBUTED APPLICATIONS.(Technology Information)

by MATTHEW STILLERMAN, CARLA MARCEAU and MAUREEN STILLMAN

Attack codes masquerading as components of distributed applications leave the application vulnerable to attacks. It is therefore necessary for applications to be able to recognize themselves, and reject imposters. This can best be done by discovering the intrusion while it is taking place. A prototype of an intrusion detection system is presented.

A distributed application is vulnerable to attack code masquerading as one of the components. An application that has been trained to recognize "itself" can reject such imposters.

The first step in defending against an information warfare attack is discovering that the attack has occurred or is in progress. Intrusion detection is the field that attempts to discover attacks (whether from external hackers or internal misuse), preferably while they are still under way [7]. Most work in intrusion detection has focused on the network or the individual computer host. Our organization, Odyssey Research Associates (ORA), is studying intrusion detection in distributed applications. In this article, we first briefly review intrusion detection and computer immunology, then describe our approach to providing intrusion detection in distributed object applications. The heart of our approach involves an empirical characterization of the application, which we call the application's "self"; here we discuss the essential components of such a characterization. We have built a prototype intrusion detection system (IDS) to protect applications [6] that are based on the Common Object Request Broker Architecture (CORBA), which is promulgated by the Object Management Group (OMG) [8]. Some of the results we have obtained from this system are described here.

Intrusion detection. Complementing traditional computer security mechanisms are relatively new tools, many in the research stage, that offer the possibility of detecting intrusions that have occurred in spite of security measures. If computer security measures are analogous to the fences and locks of the physical world, then intrusion detection is like a burglar alarm system. An intrusion detection system (IDS) might alert a human operator to a suspected intruder or might take some immediate action (such as disconnecting part of a network) to prevent damage.

One might think that the best characterization of the success of an intrusion detection system is the fraction of intrusions that it correctly recognizes, the detection efficiency. However, of equal or greater importance is the rate at which the IDS raises false alarms (the fraction of normal behavior that is incorrectly labeled as intrusive). The success of an intrusion detection system can thus be characterized by both false alarm rate and its detection efficiency.

There are two styles of intrusion detection: pattern-based and anomaly-based. Pattern-based systems are explicitly programmed to detect certain known kinds of attack. Commercially available virus detection programs are a familiar and successful example of pattern-based intrusion detection. There are also several commercial intrusion detection systems for networks that recognize well-known intrusions. While pattern-based systems tend to have a low rate of false alarms, they do have limitations. They cannot detect novel attacks; their complexity grows as the number of well-known attacks grows, introducing problems of scale; and it is difficult to keep them updated as the catalog of attacks grows.

Anomaly-based systems address these problems by attempting to characterize normal operation and to detect any deviation from normal. The challenge in such systems is to define "normal" in a way that minimizes the false alarm rate and maximizes the detection efficiency. It is interesting to note that vertebrate immune systems use the anomaly-based approach to detect intrusions; they discover "self" empirically, and attack any cells exhibiting proteins that are not self. Inspired by immunological mechanisms, Stephanie Forrest at the University of New Mexico has developed an empirical sense of "self" for privileged Unix applications, such as sendmail and lpr. Forrest has succeeded in detecting classic attacks on those applications (see [1, 2]) and more recently on the domain name service (DNS).(1) Our approach to intrusion detection for CORBA applications is based on Forrest's work. Other research seeks to characterize user behavior as normal or anomalous--see [3] in this issue.

Distributed object applications. Distributed object platforms enable developers to design an application as a dynamic set of objects whose interactions are independent of locality, language of implementation, operating system, and hardware platform. The distributed objects communicate by means of common messaging middleware, such as the OMG's CORBA or the Distributed

INTRUSION DETECTION FOR DISTRIBUTED APPLICATIONS.(Technology Information)

Component Object Model (DCOM). Distributed object platforms encourage fragmentation of the application into objects to achieve greater flexibility and other advantages. Yet each of the objects that comprise the application exposes an external interface that might be the target of an attack.

Even as it creates novel avenues of attack, however, the world of distributed object systems also creates new opportunities for integrated defense against attackers. Combining application level information about the interactions between objects with information at the operating system and network levels, we may obtain a more complete defense than is possible with only lower level information.

Application level intrusion detection systems offer an excellent opportunity for discovering misuse attacks--insider attacks that attempt to subvert an application. Such attacks can be especially insidious since the attacker is typically familiar with the application and security controls and has the best opportunity to subvert them. Many violations of security result from legitimate users performing unauthorized actions. An insider can even mount a rogue client attack, by sending messages to the application server from a client of his own design.

An application could, of course, include various tests and checks to try to detect attacks, but it is preferable to supply capabilities at the platform level that do not require explicit programming support. ORA has built a prototype CORBA Immune System to detect intrusions into and misuse of distributed object applications, based on the empirical approach to defining normal behavior. Because we define an application's "self" in terms of the middleware concepts that underlie distributed objects, our approach applies to any CORBA application. In fact, the principles are generally applicable to middleware that supports distributed object systems.

Definition of Self

Using the immunological approach to intrusion detection, we characterize "self," testing possible intruders against that characterization. The characterization is developed empirically, by observing the specific application in operation under typical conditions. The following sections explain the method of doing that and describe its use in the CORBA Immune System. Our definition of self comprises the choices we make in each of four categories:

* Focus. Which entity's self do we try to characterize? What elements or parts of the entity can be either normal or anomalous?

* Discriminating data. What aspects of the entity discriminate between normal and anomalous? For example, handwriting samples discriminate between people.

* Signature of self. What abstraction of the discriminating data characterizes the entity (as the forms of letters characterize a person's handwriting)?

* Detection algorithm. How do we compare a given entity with self? This is analogous to the problem of deciding whether two handwriting samples or two fingerprints are sufficiently similar to be deemed a match.

Focus, organism and intruder. The organism is the entity that is vulnerable to intrusion and that we seek to protect. It consists of parts, which we will refer to as cells. In this context, an intruder is a cell that appears to be a legitimate and intact part of the organism, but is not. The intrusion detection system for the organism focuses on each cell, ascertaining whether it is normal (that is, benign) or anomalous.

For example, Forrest's method detects classic buffer overflow attacks on privileged Unix processes running programs such as sendmail, lpr, and named. When successful, the attacker runs arbitrary code (such as a shell) with the privilege (root privilege). Forrest examines the behavior of specific processes that appear to be executing the privileged program to see if they are behaving "normally." The organism, in this case, is the privileged program together with all of its processes, in its customary use at the particular site or installation. The cells of the organism are the processes that execute the privileged program.

The organism we seek to protect is the CORBA application and the cells are the clients that appear to be part of the application. That is, any client of an application object is a candidate for observation. The kinds of attack we hope to be able to detect include:

* Circumventing program logic embedded in the client by breaking or abusing the client program, or by using a different program that masquerades as a legitimate client.

* Violations of trust, in which the user operates the client in ways beyond his authorization (and are unusual), but are not mechanically prevented.

The choice of organism corresponds roughly to which kinds of intrusion are of interest. For instance, Forrest focuses on privileged Unix processes, because attacks on Unix systems often occur by subverting such processes.

INTRUSION DETECTION FOR DISTRIBUTED APPLICATIONS.(Technology Information)

Our choice of CORBA applications enables us to detect many kinds of misuse, but not attacks in which the implementation of an object is maliciously modified, or attacks on the ORB itself.

Discriminating data. A school nurse decides whether a student is sick by taking the child's temperature. An immune system T cell checks the shape of a peptide. A store clerk checks the signature on a credit card slip. The temperature, the peptide shape, and the handwritten signature thus serve as discriminants between normal and anomalous. Similarly, the intrusion detection system measures some aspect of a cell to decide whether it is normal or not. In order for us to do this, three conditions must hold: there must be such a discriminant; the system must be able to measure it; and we must know which values of the discriminant are "normal."

What data can we use to characterize the self of the entity on which we are focusing? The immune system uses fragments of proteins called peptides, which are presented on the surface of cells. Analogously, in computer systems we are concerned about the behavior of entities at their interface on their externally visible behavior. We observe the behavior from some vantage point. For example, Forrest characterizes a Unix process by the sequence of calls it makes on the Unix kernel.

Note that in those experiments, the privileged processes are not attacked through the system-call interface. Rather, the system calls are a side effect of the attack, just as fingerprints are a side effect of a burglar's presence. System calls are a good choice because they capture the important effects of process behavior. CORBA applications provide several reasonable choices for monitoring. As mentioned earlier, this relatively new way of building applications exposes more of their internal workings than is exposed in traditional monolithic programs. CORBA applications typically consist of several interacting parts (the objects and clients) whose interactions, in the form of request and reply messages, are mediated by the ORB. We can observe the message traffic at the clients, at the target objects, or possibly somewhere in between ("in the ORB"). In our experiments, we monitor client behavior by observing the client's request messages as they arrive at the server (our vantage point) of the target object. CORBA interceptors, and Orbix filters in particular, provide a convenient way of obtaining this information.

In addition to a discriminant (client/server traffic) and a way to obtain it (filters and interceptors), we need to find out what values of the discriminant are "normal." A fundamental premise of our method is that we can measure discriminating data about the cells of the organism during a training period, during which we assume that no intrusion occurs. The resulting training data is summarized in a self database. During "live" operation of the system, similar data is collected for each cell and continuously compared with the normal data. If the comparison shows a sufficient disparity, then the cell is deemed anomalous and we suspect an intrusion.

Signature of self. When searching for a match to a given fingerprint in a large collection, it is useful to construct a compact description of the fingerprint in a standard way. Such a description is readily compared with similar descriptions of the elements of the collection. The description captures the essential elements of the fingerprint, while ignoring inessential variability. In some sense, the compact description defines what it means to compare two fingerprints. A good description makes it likely that fingerprints from the same finger will be identified, and makes comparison more efficient and more precise.

Analogously, our intrusion detection system must compare the message traffic on a connection between a client and a server with similar training data in the self database. We employ a short descriptor of such traffic, called a signature, that is the basis of comparison. Signatures enable the IDS to generalize from the message traffic on a client/server connection during training, to similar "live" traffic (with the same signature) that will also be declared normal.

In general, the data collected from each cell results in one or more signatures. The self database consists of all of the unique signatures extracted from the training data. Signatures of a cell during live operation are compared with the self database.

Our current experiments with CORBA applications are based on the following signature definition, which is computed in two stages. First, the sequence of requested methods is extracted from the sequence of request messages that are sent on a connection. only the identity of the methods and their sequence is preserved--all other aspects of the requests, such as arguments, are ignored. Second, the algorithm selects fixed-length subsequences of consecutive methods from the sequence (for example, sequences of four consecutive requests)--these are the signatures. A sliding window algorithm [1] populates the self database with all of the signatures present in the training data.

Our choice of the projection that forms signatures must balance detection efficiency against computational efficiency. If we discard too much information, for example

INTRUSION DETECTION FOR DISTRIBUTED APPLICATIONS.(Technology Information)

by choosing extremely short subsequences, then nearly every possible signature will be in the self database. On the other hand, keeping too much information in the signature may make comparisons too time-consuming. In general, Forrest's group has found that with the right choice of discriminating data, even projections that discard a great deal of information can provide adequate detection.

Ideally, all signatures of the running application under normal use are found in the self database. In other words, the signatures of a modest sample of training data "covers" actual normal data. In practice, we approximate coverage by collecting training data until the self database stops growing. With such coverage, it seems reasonable that the signature of cells during actual use will typically be found in the self database; this is largely borne out in Forrest's experiments. However, the convergence implied by coverage can only be achieved practically if signatures are small enough. Note that if we run the intrusion detection system with a self database that does not have coverage, we can expect large numbers of false alarms. Thus, the choice of the signature-forming projection is the subject of tradeoffs between detection efficiency and practicality of coverage.

Detection. There still remains the problem of deciding in near real time whether a cell is anomalous. That is, should the IDS raise an alarm about a particular Unix process or CORBA client? If every signature that can possibly occur in normal behavior were guaranteed to be in the self database, this would be a straightforward task. We would test each signature of a CORBA client in the running system and if any is not in the self database, the cell is anomalous. In practice, things are not that simple, because the self database is merely an approximation of normal behavior. Fortunately, normal cells appear to deviate only slightly from the self database, while abnormal cells differ markedly. We therefore need to measure the degree of deviation for each cell.

This is done in two stages: deciding for each signature whether or not it is anomalous, and then aggregating those results for all units collected from the cell in question. As mentioned earlier, the self database contains all of the signatures of all cells collected during the training period. During live operation, the IDS detector compares each new signature with the database. Anomalous signatures are those that are not found in the database; all others are normal. We have devised a finite state machine to efficiently identify all anomalous subsequences in a sequence of requests, using the sliding window technique.

We can expect that during actual operation most cells will produce many normal signatures and some anomalous ones, since even non-attack data contains isolated anomalies due to the approximate nature of coverage. Using an instantaneous measure of the cell's anomalousness together with a threshold of acceptable derivation enables a convenient way to vary the sensitivity of the IDS. The approach that we are pursuing is to base the anomaly measure on the bunching of anomalies (from one client). One would initially expect that isolated anomalies are the result of poor coverage, while concentrated bursts of anomalies signal an attack. This is borne out in practice.

Experimental Results

We tested the CORBA Immune System in two experiments. The first experiment was with LPA Vision [5], a large distributed CORBA application for parts planning. LPA Vision is widely used and controls access to potentially sensitive information. To ensure confidentiality and integrity, it implements security measures such as authentication, access control, and accountability.

Unfortunately, we were not able to obtain data from actual installations of LPA Vision, and our own resources were too limited to adequately simulate actual use of the application. We therefore decided, for experimental purposes, to build a small application that, like LPA Vision, also enabled multiple clients to access a central database.

The application we used in the second experiment is called the Personnel Tracker. The design of the Personnel Tracker follows a common pattern for CORBA applications such as LPA Vision. A CORBA server maintains a central database that associates information with its users. A per-user client provides a graphical user interface and performs password authentication. The application client enforces various reasonable access control rules.

We ran the Personnel Tracker for a month to collect training data, generated a self database, and then challenged it with simulated rogue client attacks. Since authentication is the responsibility of the client, the Personnel Tracker depends on its client to restrict access to the database. The rogue client simply allows its user unrestricted access to the data. (This is admittedly a simplistic example, although in fact, many CORBA applications use this division of responsibility between client and server to avoid authentication checks for every server request.)

Figure 1 shows the growth and convergence of the self database as training data is collected (the small "bump" at the right side of the graph represents a very small amount

INTRUSION DETECTION FOR DISTRIBUTED APPLICATIONS.(Technology Information)

of novel behavior). After training was completed, we tested the system by logging in and performing normal activities. Some of these "normal" sessions were completely covered by the training data. As we expected, a few sessions contained scattered anomalies; a profile of one such session can be seen in Figure 2.

[Figures 1-2 ILLUSTRATION OMITTED]

We then constructed two rogue clients to attack the application. The first rogue client simulates a login but does not enforce access control. Instead it gets a list of all users and alters or deletes their data.

The anomaly graph for the rogue client (Figure 3) shows very strong "bunching" of anomalies, which the CORBA Immune System was immediately able to detect and flag as anomalous. The second rogue client simply provides an interface through which the user can issue arbitrary requests to the server. The CORBA Immune System was also able consistently to detect attacks by the second rogue client.(2)

[Figure 3 ILLUSTRATION OMITTED]

We originally used a sliding window of length six, and later experimented with shorter window lengths. Somewhat to our surprise, a window length of two was just as good as a window length of six in detecting attacks. This would seem to support Forrest's observation that very simple signatures are sufficient. On the other hand, this data comes from one very simple application and is necessarily quite preliminary.

Conclusions

The current trend in computing toward open systems, exemplified by CORBA, presents new opportunities for intrusion detection due to exposed and well-characterized internal application interfaces. Intrusion detection systems for such applications--aimed at detecting intrusions into the specific application--can complement other "systemic" intrusion detection measures. Of course, such application-level intrusion detection measures could have been built into individual monolithic applications by hand. We can now envision general mechanisms that accomplish this without explicit programming. Our current work takes a step in that direction.

Intrusion detection at the application level is different for each application. An anomaly detection algorithm that derives the standard of normality can be applied empirically to each application separately. The definition of self is the rule for how to do this. This relatively new approach does not require any information about previously seen attacks and it is inherently scalable.

To test our current and future hypotheses for "self," we have built a prototype system that collects training data and subsequently uses the data to detect intrusions. This system will also help us to address the crucial question of how to make the intrusion detection system as invisible as possible to application developers and installers. A system that imposes an onerously large burden on developers and installers will simply not be used.

We have only begun to explore the immune system analogy in computers. Vertebrate immune systems not only detect infections, they respond to them and remember them. What is more, the detection, response, and memory is the collective behavior of a great many interacting agents and is thus not sensitive to the demise of any one of them. The resulting defenses are robust and efficient. Biological immune systems have achieved their current form through evolution-reproductive fitness empirically achieves a balance between costs and benefits for the organism. Can we employ fine-grained parallelism, achieving defenses as a robust emergent property? Can we incorporate memory and response? Can we empirically and dynamically arrive at an optimal tradeoff between detection efficiency, false alarm rate, and effort? We believe that computer defenses are beginning their evolution to more complex forms that will enjoy many of the advantages of their biological counterparts.

Some CORBA Terminology.

The following concepts are illustrated in the figure appearing here:

Object, An encapsulated combination of data and functionality; the fundamental abstraction of object-oriented programming.

Server. A process that manages one or more related objects.

Client, A CORBA object or any program that calls (or invokes) methods of an object.

Request. Abstractly, an invocation of a method of an object, as declared in the object's interface. Also, the concrete message conveying that invocation from client to server.

ORB. (Object Request Broker) The middleware layer that mediates the transfer of messages (requests and replies) between a client and an object.

INTRUSION DETECTION FOR DISTRIBUTED APPLICATIONS.(Technology Information)

Interceptor. A program that gains control of a message (request or reply) within the ORB, between a client and an object. Interceptors are used to transform, block, or react to the message for application-defined access control checking, and other housekeeping functions.

Filter, A form of interceptor provided by the Orbix ORB [4].

[ILLUSTRATION OMITTED]

(1) Known attacks provided a test for Forrest's detector, which does not incorporate knowledge of specific attacks.

(2) This second type of rogue client is easy to construct systematically from the Interface Definition Language (IDL) description of a CORBA server. In fact, such "clients" are typically constructed by the developers for testing and debugging the server.

REFERENCES

[1.] Forrest, S., Hofmeyr, S.A., and Somayaji, A. Computer immunology. In Commun. ACM 40, 10 (Oct. 1997), 88-96.

[2.] Forrest, S., Somayaji, A., and Longstaff, T. A sense of self for Unix processes. In Proceedings of the IEEE Symposium on Computer Security and Privacy, Oakland, Calif., IEEE Press, 1996.

[3.] Goan, T. A cop on the beat: Collecting and appraising intrusion evidence. Commun. ACM 42, 7 (July 1999).

[4.] Iona Technologies PLC. Orbix Programmer's Guide, 1997.

[5.] LPA Software. LPA Vision User Manual, 1998.

[6.] Marceau, C. et al. Architecture of a CORBA immune system. Odyssey Research Associates Technical Report TM-98-0005, 1998.

[7.] Mukherjee, B., Heberlein, L.T., and Levitt, K.N. Network intrusion detection. In IEEE Network, 8 (May-June 1994), 26-41.

[8.] Object Management Group. The Common Object Request Broker: Architecture and Specification (CORBA), 1998.

This work is supported by DARPA contract F30602-97-C-0216, administered by U.S. Air Force Research Lab, Rome, NY site.

MATTHEW STILLERMAN (matt@oracorp.com) is a principal scientist with Odyssey Research Associates in Ithaca, NY.

CARLA MARCEAU (carla@oracorp.com) is a senior principal scientist with Odyssey Research Associates in Ithaca, NY.

MAUREEN STILLMAN (maureen@oracorp.com) is the director of Odyssey Research Associates in Ithaca, NY.