Conf 9311105 -1

UCRL-JC-115015
PREPRINT

Analysis of an Algorithm for Distributed Recognition and Accountability

Calvin Ko, Deborah A. Frincke, Terrence Goan, Jr.,
L. Todd Heberlein, Karl Levitt, Biswanath Mukherjee,
Christopher Wee

This paper was prepared for submittal to the First ACM Conference on
Computer and Communications Security
Fairfax, VA
November 3-5, 1993

August 1993

Lawrence Livermore National Laboratory

ANALYSIS OF AN ALGORITHM FOR DISTRIBUTED RECOGNITION AND ACCOUNTABILITY

Calvin Ko Deborah A. Frincke Terrence Goan, Jr.
L. Todd Heberlein Karl Levitt Biswanath Mukherjee Christopher Wee

Department of Computer Science
University of California, Davis
Davis, CA 95616

Abstract

Computer and network systems are vulnerable to attacks. Abandoning the existing huge infrastructure of possibly-insecure computer and network systems is impossible, and replacing them by totally secure systems may not be feasible or cost effective. A common element in many attacks is that a single user will often attempt to intrude upon multiple resources throughout a network. Detecting the attack can become significantly easier by compiling and integrating evidence of such intrusion attempts across the network rather than attempting to assess the situation from the vantage point of only a single host. To solve this problem, we suggest an approach for distributed recognition and accountability (DRA), which consists of algorithms which "process", at a central location, distributed and asynchronous "reports" generated by computers (or a subset thereof) throughout the network. Our highest-priority objectives are to observe ways by which an individual moves around in a network of computers, including changing user names to possibly hide his/her true identity, and to associate all activities of multiple instances of the same individual to the same network-wide user. We present the DRA algorithm and a sketch of its proof under an initial set of simplifying albeit realistic assumptions. Later, we relax these assumptions to accommodate pragmatic aspects such as missing or delayed "reports", clock skew, tampered "reports", etc. We believe that such algorithms will have widespread applications in the future, particularly in intrusion-detection systems.

1 Introduction

Most computer systems have some kind of security flaw that may allow outsiders (or legitimate users) to gain unauthorized access to sensitive information. In most cases, it is not practical to replace such a flawed system with a new, more secure system. It is also very difficult, if not impossible, to develop a completely-secure system. Even a secure system is vulnerable to insiders misusing their privileges, or improper operating practices. While many existing systems may be designed to prevent specific types of attacks, other methods to gain unauthorized access may still be possible. Due to the tremendous investment already made into the existing infrastructure of "open" (and possibly insecure) communication networks, it is infeasible to deploy new, secure, and possibly "closed" networks. Since the event of an attack should be considered inevitable, there is a tremendous need for mechanisms that can detect outsiders attempting to gain entry into a system, that can detect insiders misusing their system privileges, and that can monitor the networks connecting all of these systems together.

A common element in many attacks (or computer intrusions) is that a single user will often attempt to intrude upon multiple resources throughout a network. Detecting the attack can become significantly easier by compiling and integrating evidence of such intrusion attempts across the network rather than attempting to assess the situation from the vantage point of only a single host. For example, an attacker may make only a single attempt at guessing a password for each host computer. Thus, from the vantage point of a host, the break-in attempts may appear to be a very normal mistake. However, by integrating these observations over multiple target hosts, it becomes clear that a single attacker is making a concerted attempt to break in somewhere, by looking for an obvious hole.

Accordingly, the goals of our present work on distributed recognition and accountability (DRA) are (1)

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

o ob:o observe the ways by which an individual moves ırounıround in a network of computers, including changng usıng user names to possibly hide his/her true identity distridistributed recognition); and (2) to associate all acivitieıivities of multiple instances of the same individual to he saıhe same network-wide user, referred to as the network dentifdentifier, NID (accountability). We assume that mechnismınisms (communication facilities) are available by which reporreports" can be sent by computers distributed across he nehe network to a centralized location. The centralized ıcilityıcility, a CLIPS-based expert system in our current imlemeılementation [SB+91a], executes the DRA algorithm to rack ırack users as they move around the network, maintainıg cong correct NIDs as stated above.

In Initially, the DRA algorithm is outlined under a set f simpf simplifying assumptions such as perfect network-wide ynchrynchronization, no loss of information (e.g., no loss of etworetwork packets associated with audit data), immediate reporteport" generation (i.e., all "reports" are in sequence), ıe nehe network connecting all host computers is an Etheret locet local area network (LAN) so that all of their network ctivitictivities can be picked up by a LAN monitor such as ıe Neıe Network Security Monitor (NSM) [HML+91], etc. ater, ater, some of the simplifying assumptions are relaxed, ıd thıd the corresponding necessary changes to the DRA goritlgorithm are discussed. Proofs of correctness are outıed taed to demonstrate that the DRA algorithm is robust ıder tıder the simplifying assumptions.

In In Section 2, we provide motivation for our work by scribscribing several network attacks that cannot easily be tectetected with the association of DRA. In Section 3, we iefly iefly describe a system architecture that implements form form of DRA that addresses some of the practical nsidensiderations raised earlier. In Section 4, we discuss a RA aRA algorithm under the simplifying assumption, and Secti Section 5, we extend this DRA algorithm to encomss prss practical considerations.

Network Attacks

gure gure 1 shows several behavior styles that are charac-istic istic of an intruder. Some of these behavior styles licatdicate an attempt to gain access to a system, while hers hers are intended to hide the intruder's identity or malı malicious behavior.

1 1 Doorknob Attack

a doca doorknob attack, the intruder's goal is to discover, d gaid gain access to, insufficiently protected hosts on a tem. tem. The intruder generally tries common account d pasd password combinations on several computers (see Figure 1, part b). If the intruder only tries a few logins on each machine (with different account names), single-host Intrusion Detection Systems (IDS) having a higher "threshold of detection" may not detect the attack. The threshold of detection is a common technique used to quantify "how bad" a particular behavior is. For example, two failed login attempts might be considered normal, but 30 failed login attempts should cause concern. If the intruder tries two such "doorknob rattles" on each of 15 machines, then, without aggregation, none of the 15 machines would consider such behavior to be serious. However, by aggregating all of the login failures, it becomes clear that an attack is underway.

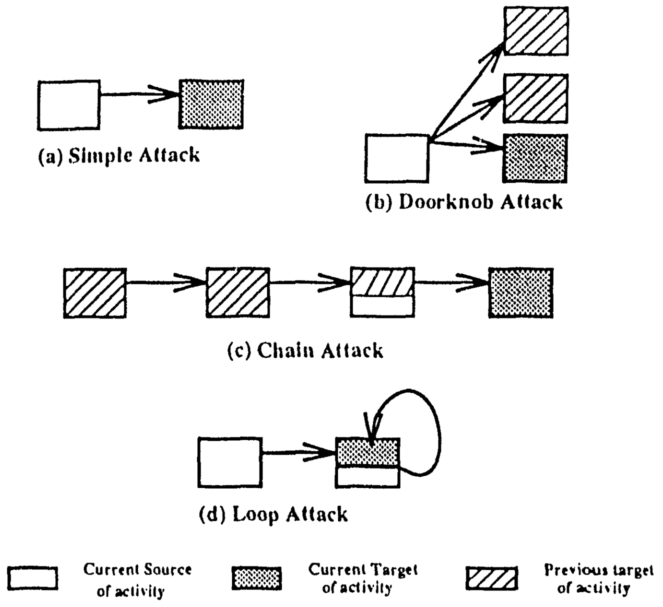
(a) Simple Attack
(b) Doorknob Attack
(c) Chain Attack
(d) Loop Attack
Current Source of activity
Current Target of activity
Previous target of activity

Figure 1: Network Attacks

For example, the following attack has been observed. An attacker gains super-user access to an external computer which did not require a password for the super-user account. The intruder uses telnet to make the connection to this site, and then repeatedly tries to gain access to several different computers at the external site.

Another intruder uses a doorknob attack and succeeds in gaining access to a computer using a "guest" account which does not require a password. Once the attacker has access to the system, he exhibites behavior which would alert most existing intrusion detection systems (e.g., writing to sensitive files). The key here is that DRA permits the intrusive activity to be associated with its original source, while most existing Intrusion Detection Systems (IDSs) would account this activity to the "guest" account and not provide any backtracking.

2.2 Chain and Loop Attacks

In a chain attack and a loop attack (Figure 1 parts c, d), an intruder moves between several hosts and account names in order to hide his point of origin. Insiders may also employ chain attacks to camouflage their identity. The key here is that the most recent reported login in a chain attack would be from an on-site location rather that from outside.

3 Architectural Overview

Our system is a prototype IDS designed to monitor user behavior across a single Ethernet LAN. It provides a type of DRA similar to that described above. Hosts attached to the LAN may be either unmonitored (such as PCs) or monitored. At present, monitored hosts must be either Unix systems with Sun Microsystem's Basic Security Module (BSM) or VMS systems. Both provide a C2-level standard of auditing information.

Our architecture combines distributed monitoring and data reduction with centralized data analysis [SB+91a][SB+91b]. This approach is unique among current intrusion detection systems. Each monitored host is provided with a Host Monitor that collects and analyzes audit records locally. These monitors pass information about notable events to a central analyzer, called the Director, for further processing. Notable events include: failed logins, changes to the security state of the system, tagged file accesses, unusually high number of file accesses (browsing), and an unusually high volume of requests for information about users (paranoia). Much of this information comes from HAYSTACK [Sma88], which has been incorporated into the Host Monitor.

The LAN is monitored by a subset of the NSM [HDL+90]. This LAN Monitor observes all traffic along the network and reports activity such as rlogin and telnet connections, security-related services, and the use of sensitive keywords (such as `passwd`) to the Director.

The Director consists of three logically independent components that are all located on the same dedicated workstation: Communications Manager, Expert System, and User Interface. The Communications Manager is responsible for the transfer of data between the Director and each of the Host and LAN Monitors. The Expert System is responsible for evaluating and reporting on the security state of the monitored system. It receives the reports from the Host and the LAN Monitors, and, based on these reports, it makes inferences about the security of each individual host, as well as the system as a whole. The Director's User Interface allows the System Security Officer (SSO) interactive access to the entire system.

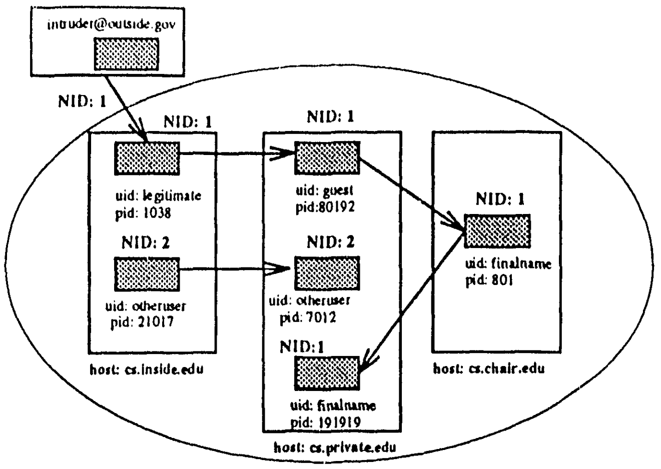

Figure 2: NID trace

4 Distributed Recognition and Accountability

Section 2 described several forms of network traversals that often indicate an intruder. Such attacks can be detected by observing the way that an individual moves around a network (distributed recognition). It is also important to assign activity to the appropriate user (accountability).

4.1 Tracking a User

On an Unix system, legal names or aliases are created for a user only upon login (from a terminal, console, or off-LAN source), upon change of user-id, or upon creation of additional aliases. In each case, there is only one initial login (network wide) from an external device and a new unique network identifier (NID) is created when this original login is detected. When a user spawns a new session, it is our goal to associate that new session with the user's original NID. Subsequent actions by that user should then be accounted to this NID regardless of the alias used.[1]

In Figure 2, a user (`intruder@outside.gov`) enters the network from 'outside'. Activity associated with NID 1 includes actions from `legitimate@cs.inside.edu`, `guest@cs.private.edu` and `final`

[1] This method only works when user creates an alias by a method which can be audited, e.g., telnet, rlogin, su, ftp. If a user succeeds in finding a method that is not recorded, a backup system (such as a LAN monitor) is needed.

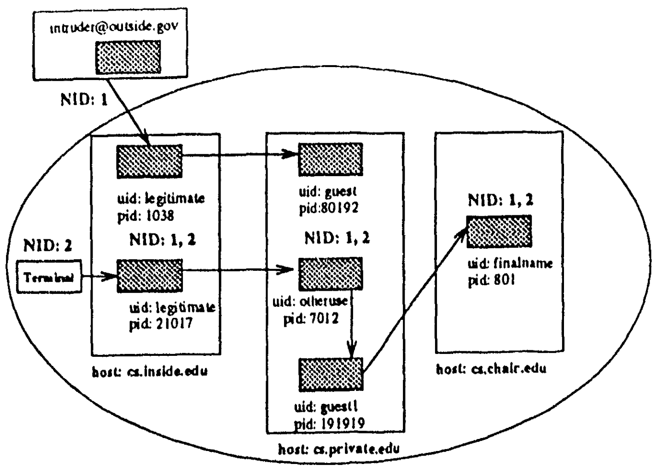

Figure 3: Tracking an Attacker

name@cs.private.edu. If the users **`guest`**, **`finalname`**, and **`legitimate`** are viewed separately, the activities associated with each of them may fall below the threshold of detection and the users would appear to be benign. If the activity associated with a particular NID is aggregated, the total may exceed the threshold of detection and it would become clear that a particular user is an intruder.

When a user creates a new alias within a local host, the local host can observe the process that initiates the alias change, and store both the original and the new alias. However, when a user creates a remote alias using rlogin, ftp, etc., the local host can observe the original alias and the request to create a new alias, but it cannot directly observe the creation of the new alias (or the process associated with it). That information is in the remote host's audit trail.

From the perspective of the destination host, the new alias (and the associated process) is known. However, the source user alias is not necessarily present. Even if this source name was attached to the request to create the new alias, the destination host could not be certain that the information is accurate. It is, therefore, necessary to combine information from both hosts in order to "connect" the activity on the source host with the activity of the destination host. This is the purpose of distributed recognition.

A slight variation of the situation depicted in Figure 3 highlights the limitation of current audit records. There are two sessions on host `cs.inside.edu` associated with user **`legitimate`**. One of them is entered from outside (`intruder@outside.gov`) and the other is logged on from a terminal. Since audit records are reported according to account name, we can do no better than account the activity of user **`legitimate`** to *NID* 1 or *NID* 2. Additional information is needed, e.g., TTY, to resolve the activity to the originating session. Of course, the IDS could identify as suspicious (but not necessarily guarantee an intrusion) the presence of two sessions with identical account names originating from different sources.

4.2 Assumptions of the DRA Algorithm

The algorithm presented here depends on the followings assumptions:

(1) all hosts on the network are monitored, i.e., they generate audit records, (2) all hosts on the network are synchronized, (3) audit records created by a host for delivery to the Director arrive in the order created, (4) no audit records are lost or tampered with, (5) an audit record identifies, where relevant, a connection identifier (in TCP/IP, the identifer is [source host and port, destination host and port]), (6) if a given user initiates multiple jobs on a single host, all subsequent activities of these jobs will be accounted as if the user had a single job (this is a limitation of current audit data in that activity is identified according to the account that generated it), and (7) each monitored host periodically sends to the Director a clock tick which indicates that all messages sent from that host before the clock tick have arrived.

The following are the message types assumed for our system. (These messages are generated by the hosts and sent to the Director for processing.)

`Connection start:`	`CS(saddr, daddr, suid, ts)`
`Connection accept:`	`CA(saddr, daddr, ts)`
`Session start:`	`SS(saddr, daddr, duid, ts)`
`Fail login:`	`FL(sadd, daddr, ts)`
`Connection end:`	`CE(saddr, daddr, ts)`
`Session end:`	`SS(saddr, daddr, duid, ts)`
`Activity record:`	`AR(host, uid, activity, ts)`

For a CS, CA, SS, FL, CE, or SS message, `saddr` and `daddr` are the source address and the destination address of the connection, respectively. The contents of these addresses depend on the transport layer used, for example, in TCP/IP, the source address and the destination address are [source host, source port] and [destination host, destination port], respectively. The field `ts` is the time stamp of the message. How these messages are generated is described below.

1. When a user logs in to a host *host_A* with uid *uid_1* from an external device *D_id*, host *host_A* will send out a SS message,

   SS(EXTERNAL, daddr, D_1, time),

where *daddr* is *(host_A, device_id)*, to the Director.

2. When *host_A* attempts to connect to *host_B*, a CS message,

   CS(saddr, daddr, suid, time),

   where *saddr* and *daddr* are the source address and the destination address of the connection, is sent from *host_A*. *suid* is the user id associated with the session which attempts the connection. (This is the first step when a user tries to creat a remote alias from *host_A* to *host_B*.)

3. When *host_B* accepts the connection, a CA message,

   CA(saddr, daddr, time),

   is sent from *host_B*. If *host_B* does not accept the connection or the attempt does not reach *host_B*, then no message will be sent.

4. If the user successfully logs in, a new session is created for him in *host_B* and a SS message,

   SS(saddr, daddr, duid, time),

   where *duid* is the user id associated with the session, is sent from *host_B*.

5. If the login attempt is unsuccessful, a FL message,

   FL(saddr, daddr, time),

   is sent by *host_B*.

6. When a notable event (e.g., deletion of a system file) occurs at *host_A*, it will send a AR message,

   AR(host_A, uid, activity, time),

   where *uid* is the user id of the user responsible for the activity.

7. When a user terminates a session in *host_C* with *uid_C*, the following activities many occur.

   Case 1: If the user started the session from an external device, a session end message SE,

   SE(EXTERNAL, (host_C, dev_id, uid_C),

   where *dev_id* is the external device associated with the session, is generated.

   Case 2: If the user started the session from another host, the following occur:

   (1) the corresponding session is terminated, and
   (2) the connection is terminated.

   When 1) occurs, a session end message,

   SE(saddr, daddr, uid_C),

   will be sent.

   When 2) occurs, a connection end message,

   CE(saddr, daddr)

   will be sent.

Note that, sometimes, when a connection is created and the user does not start a session or if the connection is closed (e.g., due to timeout or user exit), only a CE message is sent when the connection closes, since there was no session start.

4.3 Overview of the DRA Algorithm

We present an overview of the DRA algorithm and its proof, a detailed description and proof of the algorithm is presented in [KFH+93].

The DRA algorithm maintains a directed graph G(V,E), a message working set (MWS), and a connection working set (CWS) throughout the execution of the system. The directed graph G(V,E) records the current connection status of the system. The MWS stores all the unresolved messages and CWS stores all the completed connections. The originating point of a login, (e.g., an external host outside the monitored domain, or an external device such as a terminal or a console) is represented by a vertex $v(src\_id)$, which is designated as an external vertex. A collection of indistinguishable sessions (all associated with the same host and the same uid) is represented by a vertex $v1(host, uid)$. When an external login occurs, an external vertex is created and an unique network identity (NID) is assigned to it. A NID set will be assigned to each vertex since more than one originating point can create a remote alias to sessions with the same id *(host,uid)*; hence audit records of the activities of these sessions cannot be distinguished. An edge $e(vi, vj, src\_addr, dst\_addr)$, a directed edge from vertex vi to vj, indicates that a user of the session represented by vi creates a remote alias to one of the sessions represented by vj. In addition, the algorithm depends on the availiabilty of the time of the most recently arrived message for each monitored host. Therefore, the algorithm keeps a log of the the most recently arrived message's time stamp from each monitored host.

Before we describe the algorithm's detail, we first describe a likely sequence of events. For simplicity, we assume that the transport layer is TCP/IP. The source address and destination address of a connection is an ordered pair *[Host, Port]*.

- An external login occurs at time t1 from a device E to host A creating a session with account *uid1*; the message *SS(EXTERNAL,[A,E],uid1,t1)* is generated. The algorithm creates the external vertex v and vertex $v1$, and associates nid1 with these vertices. (See Figure 4a.)

- From host A port p1 at time $t2 > t1$, a connection is initiated to host B at port p2, producing the

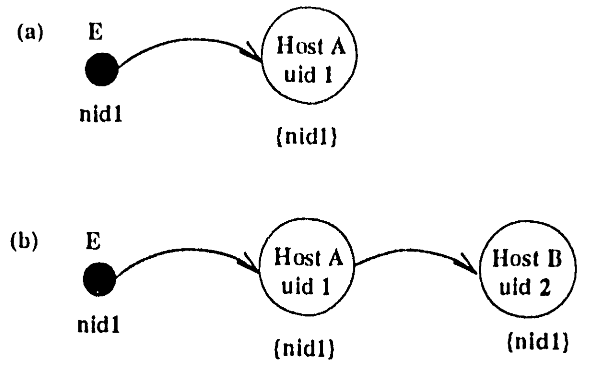

Figure 4: Normal Sequence of Events

message *CS([A,P1],[B,P2],uid1,t2)*; this message is stored in the MWS awaiting subsequent messages.

- Now, a connection accept message arrives from host B - *CA([A,P1],[B,P2],t3)*, where $t3 > t2$. The algorithm generates the pair (CS, CA) with appropriate arguments and store it in the CWS.

- How can a session be associated with this connection? The execution of a session start on host B will produce the message *SS([A,P1],[B,P2],uid2,t4)*, where *uid2* is the account name of the new session which start at $t4 > t3$. The graph G is updated to include vertex $v2$ to reflect this new session. *nid*1 is associated with the new vertex $v2$. (See Figure 4b.)

- Activitiy *AR(B, uid2, command, t5)* generated by *uid*2 on host B might now occur, and will be associated with *nid*1.

- *Uid*2 on host B terminates his session, generates the message *SE([A,P1],[B,P2],uid2,t6)*, causing vertex $v2$ to be removed. The following connection end message is subsequently generated: *CE([A,P1],[B,P2],t7)*, which removes the (CS, CA) pair from the CWS.

- Finally, *uid*1 on host A terminates his session, producing the message *SE(EXTERNAL,[A,E],uid1, t8)*, which removes vertices $v1$ and v from G.

A more elaborate situation depicted in Figure 5 illustrates the case when a vertex may represent several sessions in a host, all having the same uid. In this figure, for example, a user associated with *nid*1 and one associated with *nid*2 both start sessions on host A with uid 1. Now, when a command is executed in host A causing an activity message associated with uid 1, we have no way to determine which originating session (*nid*1 or

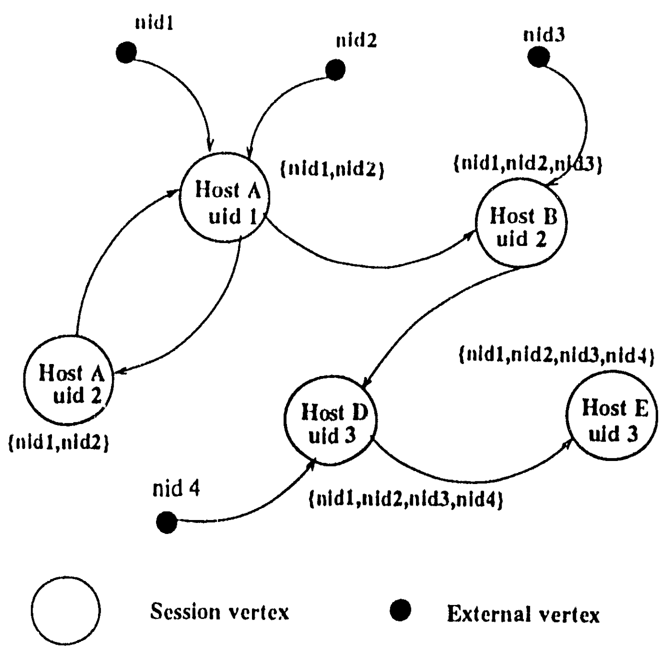

Figure 5: DRA Connection Graph

*nid*2) executed the command. Therefore, the *NID* set associated with vertex $v1$ contains the *NID* set $\{nid1, nid2\}$.

Other factors that complicate the algorithm follow:

- Although messages from a host arrive at the Director in the order generated, messages can be arbitrarily delayed (possibly due to network failure or caching of message in a host). Thus, for example, a CA message might arrive before its corresponding CS message, and the pairing of these messages must await the arrival of both messages. Furthermore, a session start followed by activities (audit events) could arrive before the CS that started the connection.

- A connection in progress might not be completed, i.e., a CS message might not be followed by a corresponding CA message. It is because the destination host does not accept the connection or the connection request is lost due to some network problems.

- A completed connection might not be followed by a successful session start. This situation happens when the connection closes (e.g., due to timeout or use exit) before the user successfully logs in.

Our algorithm reflects these possibilities. Before describing the algorithm, we indicate the properties which the algorithm must satisfy.

(1) Correct association of Activity Records: If an activity record AR is initiated by a user u from an originating session with nid n1, then n1 will be in the (NID set) associated to AR.

(2) NID sets are minimal: All $NIDs$ the algorithm associates with an activity record could have been responsible for the activity.

The DRA algorithm proceeds as follows. It consists of three major steps:

Step 1: The Director updates its log at each hosts's time upon receipt of a clock tick from the host.

Step 2: The Director attempts to pair up CA and CS messages to identify a connection when the its time log of a host H is updated. Select a relevant CA message, CA', (source host of the message is H) with the earliest time stamp. If the time stamp of CA' is earlier than H's time, pair CA' with a CS with the same connection identifier (source host, source port, destination host, destination port). If there is more than one CS message, choose the CS message that occurred most recently before CA'. Note that the pairing must await the occurrence of this CS. When the pairing is complete, earlier CS messages with the same connection identifies as the CA that did not lead to a connection accept can be discarded from the MWS. Once the completed connection is recorded, there might be SS, AR, SE messages in the MWS that can be processed.

Step 3: Process any messages that arrive. This step involves the following cases:

Case 3.1 Arrival of *SS(EXTERNAL, [host,devid], uid, time)* message. This corresponds to a login from a terminal or console. The external vertex v, corresponding to the external device is created. If there is no vertex correspond to host and user uid, create one and give it an unique NID. Otherwise, there is such a vertex $v1$ already; so create an edge between vertices v and $v1$ and add the new NID to the NID set already associated with vertex $v1$.

Case 3.2 Arrival of a CS message, which is inserted into the MWS.

Case 3.3 Arrival of a *CA([src_host,src_port], [dst_host, dst_port], uid, time)* message. Attempt to find a matching CS for this CA according to Step 2. If no match is found (the appropriate CS message is yet to arrive), store CA in the MWS.

Case 3.4 Arrival of a *SS([src_host,src_port], [dst_host, dst_port], uid, time)* message. Attempt to identity a pair (CS,CA) that has the same connection identification (src_host, src_port, dst_host, dst_port) as the SS. If there is no vertex $v1$ (*dst_host, uid*) in G, create it and create an edge from a *vi*(*src_host*, *CS.uid*) (*CS.suid* is the suid field in the connection start message CS) which must be already in G; associate with $v1$ the NID set of vi. If such a vertex $v1$ already exists, create an edge for vertex vi to this $v1$ and form the union of the NID set of vi and the NID set of $v1$. If no (CS,CA) pair has been formed, add the SS message to the MWS.

Case 3.5 Arrival of a *SE(EXTERNAL, [host,dev_id], uid, time)* message signalling the end of a session initiated from an external device. The source vertex v corresponding to the external device is removed. Also remove the vertex $v1$ if no other edge is connected to it; otherwise, update the NID set associated with vertex $v1$ to reflect the removal of the NID associate with vertex v.

Case 3.6 Arrival of a *SE([src_host,src_port], [dest_host, dest_port], uid, time)* message signalling the end of a session not initiated from an external device. If there is no vertex $v1$ in G corresponding to dst_host and uid, enter SE into the MWS, i.e., the connection has yet to be paired. Otherwise, if vertex $v1$ has only a single edge from another vertex vi, remove the edge and vertex $v1$; otherwise, update the NID set of vertex $v1$ to reflect the removal of the NID associated with the terminated session.

Case 3.7 Arrival of a *CE([src_host,src_port], [dst_host, dst_port], time)* message signalling the termination of a connection. There must be a pair (CS,CA) with the same connection identifier as CE; remove it from the CWS.

Case 3.8 Arrival of a *AR(host, uid, activity, time)* message signalling an audited event. If there exists a vertex $v1$ in G corresponding to host and uid, associate AR with the NID set of $v1$. Otherwise, insert AR into the MWS; the connection that preceded this AR has yet to be recorded by the Director.

4.4 Proof of the DRA Algorithm

The proof of the DRA algorithm proceeds in two steps. In the first step, the "major" states associated with the algorithm are enumerated, and it is proved that there are no other such states; the proof is by structural induction: for each state in the enumeration, it is verified that the arrival of a message of any type causes a transition to one of these states. The states in the enumeration (slightly approximated) include the following. They correspond to messages in the MWS with the same connection identifiers and are yet to be paired up by the Director. They are arranged in increasing order of their time stamps.

(0) Empty

(1) A single CA message

(2) n (n=1,...) CA, CE messages followed by a CA message (e.g., $CA_1CE_1CA_2CE_2CA_3$)

(3) n (n=1,...) CS messages

(4) n (n=1,...) CS messages, m (m=0,1,...) CA,CE messages followed by a CA message (e.g., $CS_1CS_2CA_1CE_1CA_2CE_2CA_3$)

As an example of the proof process, assume that the current state is (0). When a CA or CS message arrives, no pairing is possible, so that the transition is to state (1) or (3). When the current state is (1), and a CS message arrives (denote this case by †), its time stamp must be earlier than that of the CA associated with state (1). It is not known if this CS message led to the CA message or if it belongs to an aborted connection, so there is no pairing yet and the transition to state (4) occurs. Once in state (4), another CS message, CS' might arrive; if the time stamp of CS' exceeds that of the first CS message, then the first CS message is paired with the CA message and both are removed from the MWS and the transition to state (2) occurs. If the time stamp of CS' is less that that of CA, the system remains in state (4). While in state (4), a clock tick with time stamp t might arrive from the source host (associated with the connection). If t exceeds the time stamp of CA, then the most recent CS is paired with CA and the earlier CS's are discarded. When the current state is (3), and a CA message arrives, its time stamp must be later than that of the CS associated with state (3). The situation will be same as (†).

Step 2 is the proof process involves showing that the conjectured properties involving association of AR records with NID sets are satisfied – in any state where such an association is effected. This proof is again by induction, this time on the length of paths in G.

Once it is established that the collection of system states can be partitioned into the four classes above, it can be proved that the CS and CA messages are correctly paired, and a subsequent SS is associated with this pair. Next, it is shown that the following major properties are established:

(1) association of activity records with the external logins (NID set), which must contain the NID of the originating session which is responsible for the activity, and

(2) the NID set contains only those external logins which could have been responsible for the activity.

First consider property (1). The proof proceeds by induction on the length of a path in G from the external vertex v (external login) to the vertex $v1$ associated with the session(s) that caused the generation of activity AR.

Base Case: An external login results in the creation of a new vertex $v1$, and associates with $v1$ the NID assigned to the source vertex; or, in the case of an existing vertex $v1$ with the host and uid of the external login, the external login results in the new NID being added to the NID set associated with $v1$. In either case, vertex $v1$ acquires the NID of the external login and any AR associated with the new session is asociated with the NID set of vertex $v1$.

Inductive Step: Assume that there exists a path terminating with a vertex $v1$ and having an associated NID set that includes the NID associated with the external login at the beginning of the path. From vertex $v1$, a new remote session is launched by starting a connection (CS), having the remote host generate a CA message and, finally, having the remote host generate a SS message. From the above analysis, the SS is correctly associated with the (CS,CA) pair. Either a new vertex vi is generated to correspond to this new session, and vi inherits the NID set of $v1$, or an existing vertex vj and its NID set extended to contain the NID set of $v1$. In either case, the activity for this new session is associated with a NID set which includes the NID of the external login.

Now consider property (2), which involves showing that the NID set of a vertex $v1$ contains only NIDs corresponding to external vertices which have paths to $v1$. The proof is by structural induction on the graph G(V,E). The DRA algorithm updates the graph only when a SS message is processed (in this case, it creates or updates a vertex) or when a SE message is processed (in this case, it removes or updates a vertex).

Base Case: The graph G(V,E) is empty (i.e., no user is on the system). The first thing that happens must be someone logging into a host, A, from an external device, D. A SS message indicates a login from external device is generated from host A. When the SS message is resolved, an external vertex $v(D)$ with a unique NID $n1$ and the vertex $v1(A, uid)$ associated with a NID set containing only $n1$ are created. Therefore, property (2) follows.

Inductive Step: Assume that the current graph G(V,E) satisfies property (2).

The algorithm changes the graph when it resolves a SS or SE message. We have the following possibilities:

1. A SS message is resolved. The algorithm creates an edge $e(v1, v2)$ from the vertex corresponding to the source session to the vertex corresponding to the destination session. (Vertex $v2$ is created if it does not exist.) Then, the algorithm updates the NID set of $v2$ to include the NIDs in the NID set of $v1$. Therefore, a NID in $v2$ is either (a) in $v2$ before the change, or (b) in $v1$. For case (a), the property follows from the inductive assumption. For case (b), the inductive assumption indicates that the NID set of $v1$ contains only NIDs associated with vertices which have paths to $v1$. Since there is an edge form $v1$ to $v2$, those vertices have paths to $v2$. Therefore, the property is satisfied.

2. A SE message is resolved (i.e., a user terminate a session). The algorithm removes the edge $e(v1, v2)$ from the vertex corresponding to the source session to the vertex corresponding to the destination session. Then, it recomputes the NID set of $v2$ without adding the NIDs in $v1$ into the set. Therefore, the property is satisfied.

5 Relaxing the DRA Assumptions

Towards a more realistic setting for DRA, we relax several of the assumptions given in the algorithm above, and describe how the DRA algorithm can possibly be modified to accommodate these changes. As we indicate, additional information is usually required (e.g., from the NSM) and heuristics are used to attempt to infer missing data or replace erroneous information supplied by the host monitors.

5.1 DRA and Unmonitored Hosts

In order to properly track users across multiple logins, the previously presented algorithm requires monitors on all hosts. Complete accountability cannot be maintained if the user passes through an unmonitored host. An attacker can simply cover his tracks by logging onto an unmonitored host and then back onto the monitored network. (See Figure 5.) However, in many environments, hosts without monitors or even audit trails are a reality, so we are working with a technology we call thumbprinting to provide some measure of accountability through unmonitored hosts [HML92].

Suppose a user u1 on host A performs a remote login to user u2 on host B, and from host B, performs a remote login to user u3 on host C. Furthermore, hosts A and C are monitored hosts, and B is unmonitored.

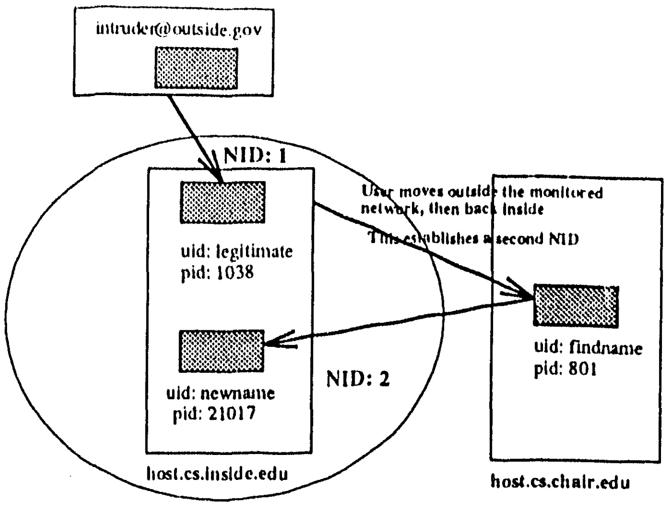

Figure 6: An Intruder Moves Off and Back Onto A Monitored Host

Our goal is to determine that the activity belonging to user u3 on host C should be mapped to the name NID as user u1 on host A. Certainly, without information from host B, we do not know to whom the connection from B to C belongs. For example, it could belong to a user logged in at the console. However, by correlating activity between the two monitored hosts, A and C, we can draw some conclusions regarding the relationship between the connection from A to B and the connection from B to C.

We start with the assumption that network connections to or from one of our monitored machines generate some monitorable activity associated with the data flow between the two machines. Furthermore, the connections from A to B and B to C are using standard login protocols (e.g., telnet, rlogin, remote shell, etc), services where a user sends data to the remote machine and the remote machine replies with information (e.g., entering a command on a command line or entering text in an editor). If these assumptions hold, we can determine, with some degree of assurance, whether user u3 on host C is really the same as user u1 on host A by using what we call thumbprints.

A thumbprint is a profile of connection activity over a specified period of time [HML92]. If two connections have similar thumbprints over several segments of time, then we can say with some amount of certainty that the two connections are really part of an extended connection. For example, we can view the two connections discussed previously, A to B and B to C, as a single extended connection from A to C. Now, we can map the activity from user u3 on host C to the same NID as user u1 on host A.

Furthermore, this technique can be extended to connections moving through an unknown number of monitored hosts, for example, $A \rightarrow B1 \rightarrow B2 \rightarrow \ldots \rightarrow Bn \rightarrow C$. By comparing the thumbprints for the connections A to B1 and Bn to C (n:=1,2,...), we can map the activity from the user on host C to the user on host A.

5.2 Out-of-Sequence Audit trails

Up to now, if a user logged on to a system and then started a process that performed a sequence of actions $a_1, a_2, \ldots a_n$ and then terminated, we have assumed that the timestamps t would be ordered as follows: log onto system$_t$ < start process$_t$ < $a_{1t} < \ldots < a_{nt}$ < end process$_t$.

However, not all auditing systems behave like this. For example, some auditing systems produce the audit records for the process before producing the audit record signalling that the process has begun: i.e., log onto system$_t$ < $a_{1t} < \ldots < a_{nt}$ < end process$_t$ < start process$_t$.

In the first case, we can use the (host) audit record to detect the creation of a new process with process id p by the user with network id n. Henceforth, all activities associated with that process id would also be accounted to NID n. Activity associated with process id p that occurs later than the timestamp of the end process action (end process$_t$) is assumed to belong to a distinct process whose process id is p only because process ids are recycled.

In the second case, we receive information about activities associated with process id p before we are able to connect that process id with NID n. Here, we cannot assign the activities to NID n until the *start process* record has been received. Further, we expect to receive exactly one record associated with process id p that occurs later than the timestamp of the end process action (namely, *start process*), and activity associated with process id p that occurs later than this timestamp is now assumed to belong to a distinct process.

Two ways to handle this problem within a heterogeneous system without modifying the audit process follow.

- Place a filter or an agent on the host that re-orders the records so that the timestamps of case 2 fulfill the order we expect in case 1. One possible implementation would require the host to record the time associated with the first observed occurrence of a process id and then set the timestamp of the start process action so that it is earlier than this time but later than the last observed action of this NID. This only solves the problem of delayed *start process* messages.
- Add information about the method used to generate auditing paths on the source host to each audit record, and use this information to group activities. This means that there must be some central location that has all of the possible audit protocols embedded in the rules.

5.3 Clock Skew

One important assumption has been that the clocks of all hosts are synchronized. In reality, this will rarely be the case.

The major problem of clock skew is that the algorithm may pair up a wrong CS message with a CA message. Consider the following scenario:

A user u1 on host A tries to login to host B, so that a CS message CS_1 is generated by host A at time t1. However, the connection attempt fails. Later, another user u2 on host A successfully logs in to host B using the same connection that user u1 used before. A CS message CS_2 is generated by A at time t2 and a CA message is generated by host B at time t3 ($t1 < t2 < t3$). Our DRA algorithm pairs the CA message with CS_2 since it is the most recent CS message before the CA message. However, if the clocks of hosts A and B are not synchronized, the timestamp of the CA message may be earlier than time t2, so that the algorithm will pair up the CA message with CS_1 instead of CS_2.

Denote the minimal time period between two consecutive connection attempts using the same connection id by t_m, and the amount of clock skew between the clocks in hosts A and B by t_c. If $t_c < t_m$, the algorithm is still correct. On the other hand, there is no easy solution if $t_c \geq t_m$. One possible way to handle this situation is to pair a CA message not only with the most recent CS before it, but with CS messages within a certain time period after it as well, depending on the amount of clock skew. However, the NID set associated with an activity record may contain extraneous NIDs.

5.4 Sharing of user id

At present, our DRA algorithm is unable to properly assign NIDs in the case of two users logging on to a second host under the same account name. The reason for this is that the Host Monitor only supplies the process id and the account name. Since the source host does not know the process id of the new process on the destination host, and the account name used is identical, it is not possible to correctly assign a NID to the new process. As indicated in the discussion on the DRA algorithm, we assign the behavior on the second host to *both* NIDs. If a more fine-grained description of the process is provided (such as the *tty* associated), it would be possible to determine which NID should be assigned the behavior. Networked PCs do not always provide even the account name associated with an activity, so this problem will appear in this type of environment as well.

6 Conclusions

We have presented an approach for distributed recognition and accountability (DRA), the purpose of which is to track users as they move from host to host in a network and to account activity to the proper user. DRA solves the problem of tracking users as they might attempt to change their identity in moving about a network. The physical environment we assume is one in which audit trails are generated at each host, and are preprocessed; and the processed data are delivered to a centralized site for analysis. The algorithm we presented makes assumptions about the hosts in a network, including: they are monitored and produce audit trails for network activity, they are synchronized, and the audit trails are not tampered with. Under these assumptions, we presented an algorithm for DRA that is proved correct with respect to a specification that asserts that activity is always accounted to the root source that initiated the activity.

We have developed an intrusion detection system DIDS (Distributed Intrusion Detection System) that has DRA as its main concept. DIDS consists of hosts each running a host monitor, a LAN monitor (NSM) that observes all network traffic, and the DIDS Director that among other things implements the DRA algorithm. DIDS was designed to work with up to 30 hosts on a single LAN segment. Our implementation keeps up with hosts generating up to 10 network activity records per second.

The paper also presents extensions to our DRA algorithm, mostly relaxing the assumptions that permitted a clean statement of the algorithm and its proof. The main extensions relate to not requiring all hosts to be monitored, to not requiring perfect synchronization of hosts, and to allowing tampering of audit records by a subverted host. In all of these cases, additional information is required to resolve erroneous reports from hosts; the NSM can in some cases provide this additional information. In addition, heuristics are employed to piece together separate chains of sessions perhaps broken by a malicious or unmonitored host.

Further extensions to DRA are under investigation. One involves extensions to much larger networks precluding the possibility of a single analysis center. Instead, the DRA analysis would itself be distributed. Other work is concerned with determining what audit data a host should collect to facilitate DRA and other intrusion detection algorithms.

7 Acknowledgements

The original concept underlying DRA was suggested by Steve Smaha who along with Che-Lin Ho, John Fisher, James Brentano, Steve Snapp, Tim Grance, Dan Teal, Doug Mansur, Ira Morrison, Bob Palasek and the authors participated in the development of an intrusion detection system for which DRA was central. A hard coded version of DRA is part of an intrusion detection system (DIDS) currently under development at Trident Systems, San Antonio. The work reported in this paper was sponsored in part by the Air Force Cryptologic Support Center and the National Security Agency's University Research Program.

References

[HDL+90] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. *Proceedings of the 1990 Symposium on Security and Privacy*, pages 296–304, May 1990.

[HML+91] L. Heberlein, B. Mukherjee, K. Levitt, G. Dias, and D. Mansur. Towards detecting intrusions in a networked environment. *Proceedng of the 14th DOE Conference on Computer Security*, 1991.

[HML92] L. Heberlein, B. Mukherjee, and K. Levitt. Internet security monitor: An intrusion-detection system for large-scale networks. *Proceedings of the 15th National Computer Security Conference*, 1992.

[KFH+93] C. Ko, D. Frincke, T. Heberlein, K. Levitt, and B. Mukherjee. An algorithm for distributed recognition and accountability. Technical report, CSE-93-7, UC Davis, November 1993.

[SB+91a] S. Snapp, J. Brentano, et al. DIDS (Distributed Intrusion Detection System)–Motivation, Architecture and An Early Prototype. *Proceedings of the 1991 National Computer Security Conference*, 1991.

[SB+91b] S. Snapp, J. Brentano, et al. A system for distributed intrusion detection. *IEEE COMPCON*, 1991.

[Sma88] S. Smaha. Haystack: An intrusion detection system. *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, 1988.

END

DATE FILMED

12/28/93