Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection Systems*

Peter Mell, Mark McLarnon
peter.mell@nist.gov, mark.mclarnon@nist.gov

National Institute of Standards and Technology
100 Bureau Dr. Stop 8930
Gaithersburg, MD 20899
8/10/99

Abstract

Distributed intrusion detection systems are especially vulnerable to attacks because the components reside at a static location and are connected together into a hierarchical structure. An attacker can disable such a system by taking out a node high in the hierarchy, thus amputating a portion of the distributed system. One solution to this problem is to cast the internal nodes in the system hierarchy as mobile agents. These mobile agents randomly move around the network such that an attacker can not locate their position. If an attacker takes out a mobile agent platform, the remaining agents estimate the location of the attacker and automatically avoid those networks. Killed agents are resurrected by a group of backups that retain all or partial state information. We are implementing this technology as an API such that existing intrusion detection systems can wrap their components as mobile agents in order to gain a type of "attack resistance".

1.0 Introduction

Intrusion detection systems (IDSs) are obvious targets for network intruders. Take out the IDS and an attacker can slip invisibly into vulnerable computer systems. This problem becomes more pronounced as commercial IDSs migrate to massively distributed hierarchical architectures. In these systems, an attacker can amputate portions of an IDS by taking out statically located command and control hosts. The resulting IDS has reduced detection capability at best and is completely disabled at worst.

This inherent weakness in modern distributed IDSs is due to their hierarchical nature. For example, if one shuts down the root node of a distributed hierarchical application, it ceases to function. However, organizing IDS components into a hierarchical structure is an ideal way to detect and respond to attacks in large networks. The majority of IDSs that scale to large networks are organized in a hierarchical fashion because this structure provides many performance and organizational advantages. The alternative, a completely distributed non-hierarchical IDS structure has been tried by several research IDSs but has proven inefficient both in detecting distributed attacks and in quickly reporting attacks.

The objective of our work is to enable IDS developers to retain the hierarchical organization of IDS components while removing the inherent vulnerabilities that hierarchical structures create. Our interest is in making IDSs resistant to denial of service attacks and penetration attacks that disable IDS components. We do not address the issues involved with an attacker penetrating an IDS and altering its functionality†. We accomplish our stated goal by converting the statically placed non-leaf components of a distributed hierarchical IDS into mobile agents (MAs).

As MAs the IDS components can hide themselves in a network, evade attackers, and recover themselves if killed. The agents randomly move around the network and thus it is difficult for an attacker to pinpoint an important agent's location. The agents are attack evasive in that if an attacker shuts down a host, agents move away from network locations they think might be under an attacker's influence. If an agent is destroyed, backup agents resurrect the destroyed agent and connect themselves back into the hierarchical IDS structure. Thus, the agent framework mitigates the single point of failure problems found

* Published in the proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection, West Layfayette, Indiana, USA; September 7-9 1999.
† However, we do limit how one penetrated IDS component can damage other IDS components.

in traditional distributed hierarchical IDSs. The attacker no longer has a statically positioned target but a mobile, evasive, and automatically recovering target.

2.0 Background on Distributed Hierarchical IDSs

Commercial IDSs are migrating towards a distributed hierarchical architecture. The architecture is a tree with a command and control system at the top, information aggregation units at the internal nodes, and operational units at the leaf nodes. The operational units can be network sniffing IDSs, host based IDSs, virus checkers, and various attack response systems.

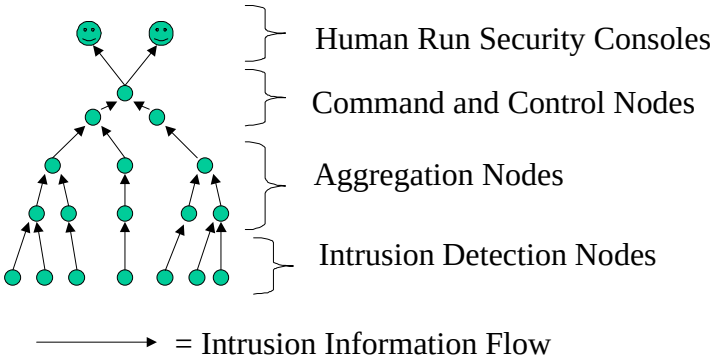

Figure 1: Generic Distributed Hierarchical Intrusion Detection Architecture

Attack detection and information gathering occurs at the leaf nodes. That information is passed to an internal node that aggregates information from multiple leaf nodes. Further aggregation, abstraction, and data reduction occurs at higher internal nodes until the root is reached. The root is a command and control system that evaluates attack situations and often issues responses. The root usually reports to human user consoles that can manually issue responses and evaluate the network.

This architecture is excellent for creating scalable distributed IDSs with central points of administration. However, the static placement of the non-leaf nodes in this architecture makes them vulnerable to attackers.

3.0 Vulnerable Systems

Most commercial IDSs and many research IDSs have this vulnerable architecture. Examples of products with distributed hierarchical architectures are: UC Davis's GrIDS [CHEN96], Lawrence Livermore's SPI-NET [SPIN99], Cisco's NetRanger [NETR], Axent's Intruder Alert [INTR], Internet Security System's RealSecure [REAL], Network Associates Incorporated's Active Security [ACTI], and Purdue's AAFID [BALA98]. It is not our intention to state that these products are vulnerable to attack. Indeed, most developers implement very tight security in their IDS nodes in order to prevent outside penetration. However, it is currently infeasible to formally prove the security of such systems and so vulnerabilities still may exist. More importantly, many denial of service attacks are effective against perfectly implemented systems. If an attacker can send the target system more information than it can handle then it will cease to function. In addition, if an attacker floods the communication channel on which an IDS node is residing, the IDS node is cut off from the rest of the virtual IDS network. These vulnerability exists regardless of the protections implemented on the IDS node. One solution to this problem is to provide IDSs a separate and protected communication channel for their operation. This solution works well but is very costly, as separate cables must be run for the IDS system. Our solution using mobile agents provides a solution to this problem without having to have separate protected communication channels for IDS nodes. However, our solution has its own set of requirements and assumptions.

4.0 Assumptions

We assume that many redundant MA platforms exist throughout the organization implementing the distributed IDS. These platforms only accept agents whose executable instructions are digitally signed by the organization's security officer. This requirement alleviates the concern that malicious agents can be

introduced into the system by compromised MA platforms. The drawback to this solution is that compromised MA platforms can still change agent's data segment causing them to lie. Agents must be carefully written such that changing the data segment can not cause them to perform malicious actions. Another problem is that compromised MA platforms may replicate valid agents thereby causing denial of service and/or confusion in the IDS. These problems exist within existing intrusion detection and response systems, however, our solution is even more susceptible since a large number of MA platforms will be installed in the organization. We do not address the problems associated with an attacker penetrating MA platforms other than to make sure that malicious agents can not be introduced into the system. Thus, the security of our system is no worse than standard distributed IDS systems where penetrated IDS components can lie and launch denial of service attacks on the other components. To reduce the threat that an MA platform is penetrated, MA platforms should be single purpose hosts accepting network traffic only to the MA platform daemon.

We assume that the network in which the IDSs are to be installed has the following topology:

- The backbone of the network is made up of a set of LAN segments connected together into an arbitrary graph.
- Cycles are permitted although in many real cases the backbone topology represents a tree with the root connecting to the Internet.
- Each backbone LAN segment has physical connections to zero or more leaf LAN segments.
- No leaf LAN segment has a physical connection to another leaf LAN segment. However, a leaf LAN segment may have multiple physical connections to backbone LAN segments as long as no traffic is ever routed through the leaf LAN segment.

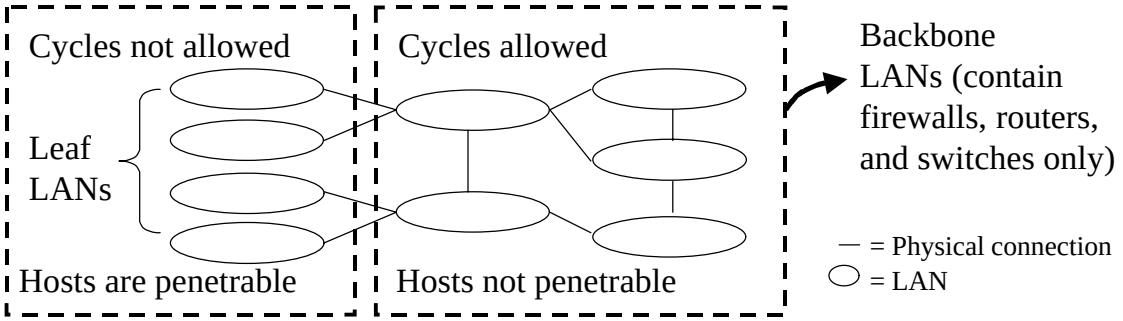

Figure 2: Example Network Topology Showing Backbone and Leaf LAN Segments

This topology is very common among corporate networks although it is not universal. Typically in this topology the computers on the backbone are routers and firewalls that are well maintained and attack resistant. We would expect these elements to fail during a denial of service attack but to resist penetration attacks. Using this assumption, an attacker would only be able to sniff traffic on individual leaf LAN segments. A sniffer at one leaf LAN segment only reveals communication involving that LAN segment since we assumed that no leaf LAN segment routes traffic to other destinations.

5.0 Our Solution: Attack Resistant Mobile Agents

Our solution provides IDS components with the ability to hide within a network, evade an attacker during an attack situation, and to recover from being killed. We do this by wrapping internal components of the hierarchical IDS as mobile agents. The internal components of the IDS can be mobile because they do not take data directly from the hosts or network. They receive data from other components, process the data, and send it onward. Thus, this processing can be performed at any location in the network (although some locations will be more efficient than others). We do not wrap the leaf components as MAs because these components typically can not be mobile and still perform their function. Host based detectors and network sniffers are responsible for particular hosts and Ethernet wires and thus have limited mobility. Future research will explore how mobility and cloneability would enhance the survivability of leaf IDS components.

Once each IDS component is wrapped as a mobile agent, we must make the agents attack resistant. First we randomize the location of all agents so that it is difficult for an attacker to target those agents which house critical IDS functionality. Next, we remove the agent's centralized directory service so that

there does not exist a statically located single point of failure that could bring down the agent system. The centralized directory service is replaced with each agent keeping track of the locations of agents with which it needs to communicate. Next, we enable agents to detect when other agents have been attacked and from where these attacks could have been launched. Agents surviving an attack evade possible attacker locations and tell other agents about the dangerous areas of the network. Backup agents resurrect agents destroyed in an attack using all or partial state information. One of many backup agents is voted the replacement for the destroyed agent. This replacement then searches the network for other agents and attempts to reinsert itself into the IDS hierarchy. Through these techniques, we enable distributed applications to be attack resistant.

We divide the techniques for implementing attack resistance into five stages, each of which will be discussed in detail:

Stage 1: Randomizing agent location
Stage 2: Removing centralized directory services
Stage 3: Evading attackers
Stage 4: Resurrecting killed agents
Stage 5: Re-establishing broken communication links

5.1 Stage 1: Randomizing agent location

Once IDS components are wrapped as MAs, randomizing their location is simple. Agents can ask their current MA platform for the list of available MA platforms and randomly choose one. Alternatively, each agent can carry around a list of MA platforms. We have agents continuously and randomly move around the network so that an attacker will have trouble pinpointing the location of critical agents. The attacker can scan a network and find all MA platforms, but it will be impossible to detect what agents are residing on each platform.

One possible avenue for attack against our system would be to monitor communication between MA platforms. We assumed that MA platforms encrypt all communication traffic. Even so, an attacker can still analyze the source and destination of the traffic as well as the amount of traffic. By doing this, an attacker can estimate how many agents are on a target platform and determine those platforms that contain agents that wish to communicate with agents on the target platform. However, since agents will be constantly and randomly moving around the network, it will be difficult for an attacker to gain a picture of the underlying IDS hierarchy by only monitoring network traffic at a few locations. Even if the attacker has sufficient sniffing points, having MA platforms send other MA platforms random garbage transmissions may fool the attacker.

5.2 Stage 2: Removing centralized directory services

An attacker, frustrated at not being able to target the location of critical IDS components, may instead attempt to disable the MA system that is acting as the foundation for the IDS. This can be done by shutting down all MA platforms in a network or by finding single points of failure on which the MA's rely. We assume that the attacker does not have the resources to shut down the majority of the MA platforms since we assumed that many redundant platforms exist. Thus, an attacker can only disable the MA system by finding a single point of failure on which the MAs depend.

MA platforms (and MAs) probably must depend on technologies such as a public key infrastructure and a centralized mobile agent directory service[*]. These services may have backups in place to provide fault tolerance but assume that an attacker can take out a small number of backups. Although it is somewhat unwieldy, the reliance on a public key infrastructure can be removed by having each MA platform keep copies of the relevant tables locally.

The need for a centralized MA directory is harder to remove. The tables for the centralized mobile agent directory service could be stored locally on each MA platform, but then when an agent moved it would have to update its entry on every platform and this is inefficient. Furthermore, an attacker sniffing the network would be updated on the movements of each agent no matter where it was physically located.

Our solution is to have each agent define a set of buddies. A buddy relationship is mutual and defines a set of two agents that will constantly notify each other of their whereabouts. When an distributed

[*] In order an MA to communicate to other agents, it must obtain the location and unique identifier for the desired agents. This location and unique identifier is most easily obtained from a centralized directory service.

hierarchical IDS begins execution, each IDS component buddies up with the other IDS components that it must communicate with in the hierarchy. Each IDS component can then move around the network at will but must notify its buddies of any change in location.

While removing the centralized directory sounds easy, it causes many difficulties most of which deal with deadlocking agents. For example, if two buddies move at the same time then they will lose each other. Thus, buddies must take turns when they move so that communication links are retained. However, if buddies wait on each other, then a group of buddies could deadlock if their pattern of waiting is a cycle. In addition, since buddies wait upon each other to move, MAs that move often and quickly may disallow a slower buddy from ever moving. We solved these problems, and others, using a concurrent negotiation algorithm that we developed. Unfortunately, there is not room in this paper to elucidate the details of the algorithm.

5.3 Stage 3: Evading attackers

An attacker may kill some agents if he takes down an MA platform. Buddies of those agents detect this event and take evasive action to avoid the same fate since their location may also be visible to the attacker. Evading agents that survived the attack avoid network locations suspected to house the attacker and inform other agents of these dangerous areas. Which network locations an agent avoids are based on how the attacker chose the target and thus where the attacker could have launched the attack. Agents must consider three cases.

Case 1: Random Choice

For case 1, assume that the attacker took down an MA platform by randomly choosing a host. In this case, agents achieve no increase in their security by moving. Assume now that the attacker scanned the network to determine the location of all MA platforms and then chose an MA platform randomly. Since agents must reside at all times on MA platforms, it again does not increase their security if they move. However, neither does moving decrease their security.

For cases 2 and 3, assume that the attacker took down the MA platform because the attacker knew that the MA platform housed important agents. An attacker can only obtain this knowledge by controlling a penetrated MA platform or by cleverly analyzing encrypted network traffic patterns. No other interfaces exist by which the attacker can discover what agents reside on an MA platform.

Case 2: Knowledge gained by controlling an MA platform

If the attacker controls a penetrated MA platform, then he knows which MA platforms are talking to the agents on his MA platform. He knows which agents recently left his platform for the target MA platform. In addition, if one of the agents on his platform is talking to an agent on another platform, he can obtain a list of LANs with which the other platform is talking[*]. These are the only way of detecting what agents are on other MA platforms because we assume that one MA platform can not ask another MA platform for a list of its agents. In addition, an attacker controlling a penetrated MA platform can not launch malicious agents to find out such information because MA platforms only accept agents digitally signed by the security officer (see section 4.0).

Case 3: Knowledge gained by sniffing network traffic

If the attacker is sniffing network traffic, then he may be able to estimate the number of agents on a particular host by watching the number of connections and amount of traffic to and from an MA platform. Our topology assumption in section 4.0 requires that:

1. No leaf LAN (as opposed to a backbone LAN) routes traffic between two other LANs.
2. All MA platforms are located in leaf LANs.
3. Backbone LANs can not be penetrated and thus the attacker can not use them to sniff network traffic.

Given these requirements, the traffic between two MAs in different LANs can only be sniffed by attackers located in the two LANs respectively.

[*] The attacker can obtain this because MA platforms distribute “dangerous LAN” lists. This feature is introduced later in this section.

The agents surviving an attack cannot specifically identify which of the three cases the attacker used to choose the target MA platform. Therefore, they evade all LANs that could house the attacker under all three cases:

Case 1: Moving does not benefit an agent although it does not hurt.

Case 2: An agent must avoid all LANs housing MA platforms that recently communicated with the target MA[*].

Case 3: An agent must avoid all LANs specified in case 2 as well as avoiding the LAN housing the target MA platform.

Agents surviving an attack can not determine the LANs that the target MA platform was communicating with after the target LAN has been taken down. Thus, they need to obtain this information ahead of time. The solution is for MA platforms to serve out a list of LANs that they are communicating with to any agent talking to the MA platform. We call this the "dangerous LAN" list. Agents talking to an MA platform periodically poll to obtain an updated dangerous LAN list. If an MA platform is taken down, agents communicating with that MA platform tell other agents about the event and distribute the "dangerous LAN" list. Entries on a "dangerous" list may time out and may have varying degrees of priority depending upon how many times a LAN has been sited as dangerous.

5.4 Stage 4: Resurrecting killed agents

In the event that an attacker kills an agent, we must resurrect the agent. The resurrected agent at minimum needs to contain the code segment of the original although ideally it contains partial or full state information. We accomplish this by having each agent clone multiple backup agents that maintain constant communication with the original. This communication link updates the backups with a degree of state information of the original and notifies the backups when the original has died (or become unavailable in which case the agent is expected to kill itself). In the event of the death of an agent, the backups negotiate among themselves as to who will take the place of the original. The chosen backup assumes this position and the rest die.

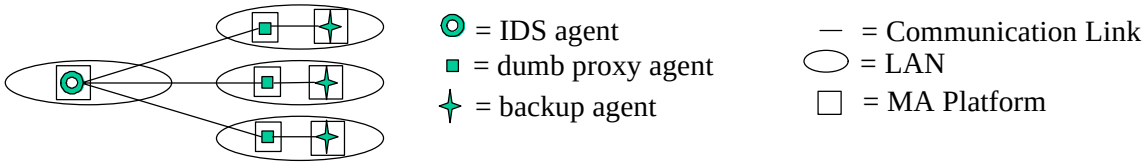

Figure 3: The Communication and Location Model for Agent Backups

While resurrecting a killed agent is straightforward, there is some complexity that must be handled. If an attacker is located in the same LAN as the original, when the attacker kills the original then the attacker will see all communication paths to the backups. Thus, an attacker can kill an agent and all of its backups simultaneously. One solution to this problem is to create more backups than the attacker is assumed able to kill. However, this solution becomes very expensive since a constant communication link must exist between the backups and the original. Our solution is to require each backup and the original to reside on a unique LAN. Each backup has a dumb proxy agent that resides on a different host in the same LAN. The backup communicates with the parent by talking to the dumb proxy agent and the proxy forwards communication to and from the original. This way, an attacker located at the original's LAN can only see the original and a set of dumb proxies. If a proxy is killed then its associated backup issues a new proxy that then tries to reconnect to the original. If the attacker is located on one of the backup's LANs, then the attacker can kill only that backup, its proxy, and the original. The original creates multiple backups with the assumption that the attacker can not have a simultaneous presence in each LAN containing a backup. The more backups created, the greater the assurance for the original and the greater the computational load that exists on the network.

If the original is killed, the backup agents must figure out who should replace the original. They do this by sending "voting" agents to a pre-agreed upon location. If this location is unavailable then they

[*] We avoid the LAN containing the penetrated MA platform because internal LAN security is usually lax and the attacker definitely has sniffing capability within the LAN.

continue down a list of agreed upon locations until a functional one is found. The voting agents do not wait for all backups to send their representatives since some backups may have been destroyed by the attacker. Thus, the first few voting agents that arrive vote on a successor to the original. They then post the result on the complete list of pre-agreed upon locations to let other backups know that a replacement has already been agreed upon. The winner agent sends a message to its backup telling it that it should assume the original's place. The other voting agents subsequently tell their associated backup agent to kill themselves. The voting agents then kill themselves.

5.5 Stage 5: Re-establishing broken communication links

When a backup agent assumes the original's position, it sends out backups of its own to protect itself. But more importantly, it must reconnect into the IDS hierarchy. This is not as simple as it sounds since there is no centralized directory of agent locations and since the agent's "buddies" may have moved on to different hosts.

There are two solutions to this problem. First, when agents leave a host they may leave a temporary marker at the first host saying that they left for the second host. A resurrected agent can look at the last known locations of its buddies and try to find these notes. It then may be able to chain from note to note until it finds the current location of the buddy. This solution provides resurrected agents a fast way to find their friends, however, it is not foolproof. If an attacker kills two buddies, then the two resurrected agents can not use this technique to find each other.

The slower, but more effective, method is for the resurrected agent to stay stationary and to probe the existing MA platforms (this list is available from the MA platform) until it finds the lost buddy. If the resurrected agent finds any agent connected to the IDS hierarchy then, through this agent, the resurrected agent can send a request through the IDS hierarchy advertising its position. This mechanism is slow and inefficient but enables agents to find each other and to be reinserted into the IDS hierarchy.

6.0 Implementation

We are prototyping this technology as an API that we plan to distribute to the public. The prototype is to be a functional version of this technology but will not implement any encryption or public key infrastructure features. IDSs will be able to use our API to render their backbone components "attack resistant". Our implementation is written using the IBM Aglet platform and Sun's Java Development Kit 1.1.8. Being Java based, it will run on any Java enabled device. At present, we have completed stages 1 and 2 as defined in section 5. Thus we have created location independent components that randomly move around the network and that do not rely on a centralized directory service. This functionality makes it much more difficult for an attacker to target specific agents and to find single points of failure in the MA system.

7.0 Conclusion

Mobile agents are an effective mechanism to provide distributed IDSs a type of "attack resistance". Since the non-event generation components of a distributed IDS can reside anywhere in the network, they can be easily wrapped as mobile agents. These IDS components, wrapped as mobile agents, then automatically evade possible attacker locations, eliminate single points of failure, and resurrect destroyed components.

8.0 References

[ACTI] Network Associates Inc., Active Security Product, http://www.nai.com/asp_set/products/tns/activesecurity/acts_intro.asp.

[BALA98] Jai Balasubramaniyan, Jose Omar, Garcia-Fernandez, E.H. Spafford, and Diego Zamboni, *Architecture for Intrusion Detection using Autonomous Agents, Department of Computer Sciences, Purdue University, Coast TR 98-05; 1998,* http://www.cerias.purdue.edu/projects/aafid.html.

[CHEN96] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, "GrIDS -- A Graph-Based Intrusion Detection System for Large Networks". *The 19th National Information Systems Security Conference, pp. 361-370, October 1998.*

[INTR] Axent, Intruder Alert, http://www.axent.com/product/smsbu/ITA/default.htm.

[NETR] Cisco, Net Ranger, http://www.cisco.com/univercd/cc/td/doc/product/iaabu/netrangr.

[SPIN] Lawrence Livermore, Security Profile Inspector for Networks, http://ciac.llnl.gov/cstc/spi/spinet.html.

[REAL] Internet Security Systems, Real Secure Product, http://www.iss.net/prod/rs.php3.

9.0 Author Information

Peter Mell is a computer security researcher at the National Institute of Standards and Technology. He has been working the field of computer security for three years. His first two years were spent developing intrusion detection systems for the Defense Advanced Research Projects Agency. During this time he obtaining a masters degree in Computer Science from the University of California at Davis. His current research interests include: mobile agents, computer attack scripts, attack script databases, and intrusion detection. His recent research and papers can be found at: http://www.itl.nist.gov/div893/staff/mell/pmhome.html.

Mark McLarnon is currently an intern with the NIST Computer Security Division. He attends the University of Maryland Baltimore County and is majoring in Computer Science, currently he is beginning his junior year. Having been with NIST for over a year, his past work has been with web based digital signature technology applications. Current research interests include: mobile agent technology, cryptographic token applications and intrusion detection.