Going Beyond Behavior-Based Intrusion Detection

Michael R. Hines

Abstract: Today's Intrusion Detection (ID) ideas focus on two solutions: detecting intrusions based on known vulnerabilities and detecting anomalies in the normal "behavior" in a particular system, be it the network or the host. Although not immune from false-positives or true-negatives, these systems do a lot for ID, but the damage from unknown attacks is more substantial as people deploy these systems and let their guards down. This paper attempts to see if a guarantee of security can be asserted in two popular ID areas: Host-Based ID and Denial of Service ID on the network. First, we explore what capability systems can do for Hosts and then explore the newer methods of Denial of Service source identification. These ideas have the potential to do much more than unreliable dependencies on behavior based Intrusion Detection.

I. Introduction

Today's proliferating Intrusion Detection solutions can be classified in many ways. They can operate in either a host or a network environment, and within both of these settings the ID systems can be classified as either a Knowledge based, or a Behavior-based system [1]. Knowledge based ID systems attempt to detect known vulnerabilities such as software or operating system holes, viruses with matching signatures, IP address spaces known to act up, and bad firewall/network/routing configurations. The list goes on (and on). If an ID system does not have knowledge of a new vulnerability, it can enumerate or build a database of normal system/network behavior and use this information to detect abnormalities. Then, maybe the ID system will find valid problems, and maybe it will not.

Both Knowledge based and Behavior based ID systems do a very good job for the most part, however when these systems detect things that are not intrusions (false positives) or fail to detect real intrusions (true negatives), they may become less effective. It is impossible for behavior and knowledge based systems to do any better. Knowledge based systems obviously cannot do more than what they were told, and simple behavior based systems are not smart enough to detect what is "new" and valid behavior without going into deeper reasoning or even artificial intelligence algorithms. In the process, they must inefficiently keep a lot of state and audit data in order to increase accuracy.

This paper will explore two important areas of Intrusion Detection: security on the host and denial of service on the network, and for each area there is an initial discussion on what the best knowledge and behavior-based methods are. Then, we explore the question in each area of whether it is possible to guarantee security and possibly detect and/or prevent intrusions that behavior/knowledge based systems cannot. Specifically, for ID on the host the idea is to use capability systems and for Denial of Service, the idea is to provide an improved solution for IP traceback, the converse of traceroute for spoofed source addresses.

II. Host-Based Intrusion

At any given moment, there are several things going on at a host: A resource causes an interrupt, doing I/O to a particular system resource like a process or physical device, which in turn does the exact same thing. Systems have so many resources to both create and accept I/O that it is nearly impossible to keep up with them all. Today's operating systems all implement generalized forms of access control [2]. All users, processes, and devices belong to a particular group in the system. In UNIX general users may have their own group ID, common output devices and storage devices may have their own group ID and administrators are in their own group. Resources that are accessed have their own set of meta-data consisting of what group can access that resource and what type of operation can be performed on it, i.e. read, write, or execute. Windows access controls are sometimes not as stringent from default installations, but they can easily be configured to provide rigorous access control. People then find ways to get access to resources that nobody initially thought about: if a user can do a particular I/O on a resource that causes *another* resource to give that user access to a resource which they did not originally have access to, then they have subverted the system. So, administrators configure their systems to be what they think is as secure as possible, and sooner or later a combination of I/Os allows a user, process, or device to gain access to a resource not originally thought possible.

This game of hide and seek is a never-ending one. How might we detect invalid access rights intrusions ahead of time? As stated earlier, either we know them ahead of time, or we look for anomalous behaviors among I/Os. One way is to keep track of system calls [3]. In order to do an I/O on any resource, you must go through the Kernel and utilize any of the thousands of system calls that it provides. If we take the knowledge based approach, we can gather a record of commonly known intrusions used to gain illegal access rights. We can then utilize this audit data to map out the sequence of system calls used in gaining access to a particular resource and modify our system to detect these sequences. However, the instant someone discovers a new combination of system calls that can be used to exploit a resource, we are out of luck. Dorothy Denning explores this problem in her paper *An Intrusion Detection Model* [4]. There, resources are described as Objects and users, or anything else that accesses those Objects, are called Subjects. In addition to her model, we must take into account that at any point in time, an Object can immediately turn into a Subject or vice versa. For instance, a process Object that is taking input from a user is also a Subject giving input to the Kernel. Similarly the Kernel, being an Object of the process Subject is also a Subject of all physical device Objects in the System. Objects or Subjects that can change in this manner always present dangerous situations, creating the possibility that a Subject can cause an Object to grant that Subject access to something it would normally not have had access. That access-granting Object is no longer just an Object, but is now a potentially harmful Subject, and it is possible to model complex graphs that represent these types of interactions. When knowledge fails, we can do audits on the system and perform statistical evaluations on the frequency and types of accesses that any particular Subject has on an Object. In essence, when we have this information, we can use those evaluations (behaviors), do the same evaluations of currently running accesses between Subjects and Objects and compare them to normal behavior. If an anomaly is detected, then additional steps can be taken to investigate, and possibly act upon it.

Furthermore, you can combine the two approaches by keeping state on Subject/Object interactions. Some intrusions may have new I/O combinations, but pieces of them may be based on the strategy from a previous intrusion. An IEEE paper describes development of a tool STAT [5] which keeps state on accesses that start with legitimate access rights and lead to illegitimate accesses. If a sequence of Subject/Object interactions matches the beginning, middle, or end of the behavior of a previously known intrusion, or if a sequence contains *many* of the same I/Os (not necessarily in the same order) that a previous intrusion does, then it is safe to assume that a resource is being accessed illegally.

It seems inevitable that all of the intrusion detection techniques mentioned thus far will determine intrusions *after* the fact. Access control can be improved upon. An operating system idea that is not very popularly implemented is capability systems [6]. Access control lists in today’s operating systems allow Subjects to gain access through I/O combinations that can lead to the illegal construction of new access control lists. The idea behind capability systems is that Subjects should *not even be capable* of doing such things; A Subject's access rights should be contained and only granted to the Subject explicitly. This is the idea behind capability systems. Although the two concepts are related, capability systems and Access Control Lists are not equivalent [6]. Capability systems attach access rights to the Subject, whereas Access Control Lists attach access rights to the Objects, such as Date-Modified and ugo+rwx meta-data in common filesystems. If the access rights are directly attached to the Subjects from the start, the Subject will not be able to access a particular resource unless explicitly given that right. In capability systems, a Subject's access rights consist of read, write, execute, and an additional right called grant. Not only does a Subject have the ability to access a resource but it can be *granted* the ability to *grant* other access rights to Subjects. This idea has actually been implemented, under the effort of Dr. Shapiro at The Johns Hopkins University. His team works on a completely capability-based operating system called EROS (Extremely Reliable Operating System) [7].

Such a system is provably secure in that it can prevent access-specific intrusions based on Subjects trying to gain access to resources that they should not be permitted to access. This is done by having the administrator grant explicit capabilities to Subjects on particular Objects where appropriate. For instance, capabilities could eliminate a problem like buffer overflows. Buffer overflows happen when the amount of input to a procedure call or variable is so large that it overflows into that process's instruction memory space. This overflow contains executable code, causing the process to execute instructions it was not originally intended to execute. If you could cause a webserver (which probably has root permission) to change or delete the Unix password file, then you have created a situation in which an Object (the webserver) turns into an illegal type of Subject, causing access rights to be changed in the system by modifying the password file. In a capability system, the webserver-process must *explicitly* be given access to the Unix password file regardless of whether or not it was owned by root. Such a system is by no means easy to configure, as one would expect. For instance, what policies do you use when deciding to whom you should grant *"grant access"* rights? If you do not choose wisely, you could compromise your system by giving a subject too much authority. However, such a system is a plausible competitor in preventing intrusions for which there is no previous Knowledge. I believe this is the only solution short of continuously playing hide and seek with other types of intrusion detection; by completely containing what a Subject is capable of, you can restrict intrusions all together. Completely capability-based systems also have a performance advantage over the competition. Because capabilities are so explicit, there is essentially no need to continuously audit the system for abnormal behaviors. A subject will never be able to access something to which it was not explicitly granted that capability. High overheads can come from watching system calls, keeping state transition data and watching over the entire system like a hawk as intrusion detection systems are often designed to do.

To keep the system provably secure, we must assume that the administrator properly performed the grueling task of carefully constructing capability setups. In some situations, this may not be an acceptable assumption to make, and as a result it may still be desirable to keep some sort of system that keeps watch over Subjects who have possibly system-compromising grant access capability rights. Thus, I believe that it may be useful to develop some sort of security verification system for the grant capability. One could imagine a graph of the system that would contain a series of nodes and edges interconnecting all the subjects and objects that have interacting read, write, execute and, most importantly, grant capabilities. The verification system would populate all the nodes in the system by searching and identifying all Subjects and Objects in the system. The verification system would then enumerate all possible edges of the graph connecting those enumerated Subjects and Objects, most importantly including the grant-capability. The system could then use this graph to discover cycles or paths in the graph that indicates the ability of a subject to acquire un-wanted grant accesses on Objects of which the administrator was not immediately aware. This verification system would not necessarily be a real-time analysis, but it could be performed when the system's capabilities are initially defined and only re-run when the administrator modifies those capabilities. This would aid the administrator to a great degree and help prevent Subjects from illegally acquiring capabilities to Objects.

III. Network-Based Intrusion

It is amazing how many problems arise when you decide to connect your machine to the network. Again, we begin with knowledge-based intrusions, first providing a brief survey of some of them. In the past and present, the first thing an attacker does is pretend to be who he is not. By utilizing the lack of security in current network layer and transport layer implementations, attackers run servers on alternate ports, forge source IP addresses, and even forge those pseudo-unique, pseudo-permanent, manufacturer-issued MAC addresses on your network interface card. Attackers also replay transmitted data, create new data, delete data, prey on bad encryption schemes and play with routing protocols. Solutions to these problems that we *know* about are obvious and immediate: prevent what you know. First, we throw up a firewall, which is a great solution until a particular software package's security problems poke a hole in it. There is no way for the firewall to know it has a hole in itself until the unfortunate occurs. Software projects like Ethereal [8] work to reach inside packet payloads and pull out protocol indicators that point to unwanted protocols. Proxy servers and router Ingress/Egress filtering can take care of *some* bad source IP and MAC address problems inside subnets. Encryption technology is actually very strong, and it can be argued that human errors resulting from protocol complexities will subside as stronger programming techniques take over and computational speed increases to eliminate mathematical limitations.

But, then there is the Denial of Service attack: bring the network down if you cannot get into it. The first most common way of doing this is by causing a surge of traffic on the network that is too great for routers and hosts to deal with. Targets are often switches, routers, or proxies, which are single points of failure or bastion hosts that directly provide remote services like websites or user-accounts. These surges can be completely generated from the resources of the attacker or by using other machines on his behalf. The latter is often accomplished by having multiple, pre-compromised machines outside of the network that simultaneously flood the target network/machine with packets (termed “Distributed” Denial of Service or DDoS). Transport-level packet choices have significant effects on how the denial of service is accomplished. For instance, an ICMP-echo based attack might be chosen because of the absolute need for some hosts to respond to ICMP-echo packets. The target machine may not be able to process echo replies as fast as echo requests are coming in. UDP packets may not be particularly interesting because there is no state associated with a UDP packet – it is just blindly sent. TCP packets have lately become of more interest because the TCP/IP stack of the kernel must be ready to create connections and take down connections based on the type of TCP packet sent. If a TCP SYN packet is sent to a machine’s open

TCP port, the machine would be forced to allocate memory, complete the three-way handshake, and keep state for that particular request until it time's out or is closed by hand. Too many SYN packets would cause the machine to eventually run out of resources [9].

Once again, we are forced to explore the unknown: what should we do if we do not already know the details of a specific attack? Due to the possibly high-volume nature of the network, we are going to want an efficient, behavior-based way of detecting DoS attacks. DoS attacks are in somewhat of a weird class of their own. A sudden increase in load from a DoS attack is not easily detectable. You are not necessarily detecting a "behavior" nor are you detecting something you already know about. The "behavior" is simply too much load on the network, and DoS attacks do not happen in a predictable manner. DoS attacks thus create two problems to solve: 1. Mitigate the load and 2. Identify the source of the attack.

Probably the least interesting of the two problems is lightening the load. There are many solutions to this problem, comprising combinations of firewalls that restrict bandwidth and specific packet types and proxies that keep state on incoming packets. Proxies, for instance, can sit in front of the bastion hosts and "take the hit" for the hosts, by forwarding SYN requests, immediately issuing RST packets to the bastion hosts in case those SYN requests are not followed up upon, a technique known as "scrubbing" [10].

Because smart DoS attacks never include valid source addresses, the more interesting thing to do is to figure out where the attack started. The first inclination one might have would be to get the source "straight from the horses mouth": i.e. ask the routers. It would be convenient if one could write a modified routing algorithm that could actually keep state on the packets that they routed, such as a modified BGP algorithm with timeouts on the amount of state they kept. Because this approach would quickly become a very inefficient use of a router’s internal resources, another suggested approach involves extracting the state out of the routers and putting it directly into the packet. The basic idea would be that as soon as a router gets ready to route a packet, it would “stamp” the packet with some meta-data that indicates, “I routed this packet.” During or after the DoS attack, you could simply look inside the packet, see which routers “stamped” the packet, and trivially trace the route the packet took through the internet all the way back to its source. In effect, it becomes the equivalent of doing a traceroute without knowing the actual address of the source. I believe such a solution (even if it is as hard to spread as IPv6) has the potential to completely eliminate all serious DoS attacks. If attackers new that it were impossible to hide from identification, they would not even think about trying to execute the attack in the first place. This would certainly not eliminate the need for load balancing though; there will always be unintentional Denial of Service cases caused by *flash crowds* [11], instances where many users intentionally (or unintentionally) and simultaneously attempt to access services on a remote network. However, attackers would most certainly think twice before executing a DoS attack.

One paper published through the ACM [12] attempted to address this very issue. There are two main questions that arise when you want to “stamp” a packet: 1. What do you stamp a packet with? and 2. Where in the packet do you *put* this information? The most intuitive answer to the first question is to have all the routers in the chain insert their IP addresses into the packet. At the end of the route, every packet would contain a complete traceroute. Containing all that information would be very costly, so they go on to say that instead of storing every router's address, just have each packet contain only a "sample" of the route. As with any node, or edge based, graph-structured system there is a graph of all the links and nodes involved in a particular route that a packet takes. Each router would "flip a coin" and based on a hard-coded probability, it would decide to actually insert the combination of its address and its distance from the source of the route into the packet. If the DoS attack generates enough packets (which it probably will in order to accomplish its purpose), chances are that every router will be the only router that marks a particular packet at least once. Using those addresses and distances, the source of all the DoS packets can be reconstructed from the samples. This can be done by either sampling nodes or edges.

This is a great idea, except where and how do we put all this node, edge, and distance information? The authors suggest utilizing the commonly un-used IP Identification field, usually used for IP packet-fragmentation. Without going into too much detail, this 16-bit field could be used to store hashes of the metadata used in the above ideas. However, making a slight modification to this algorithm would greatly reduce the number if bits of metadata needed to be stored in the 16-bit field. Storing addresses requires 32-bits of data, and if you do edge-sampling you need 64 bits. Is not there some other way to identify the route a packet takes? Yes, let us use the Autonomous System numbers. The internet is composed of many large Autonomous Systems. They are groups of systems all completely independent in and of themselves. Autonomous Systems talk to each other through BGP and thus need pre-assigned numbers to identify themselves. If an AS could stamp packets with their AS numbers, this would still give us quite vital information on the route of a packet, and law enforcement could greatly reduce the size of their investigation by knowing the exact AS source from which the attacker launched his attack. AS numbers are no larger than 16 bits, and this would make perfect use of the IP fragmentation field. Furthermore, if we continued to take a hash of the 16 bit AS numbers, we could even include the distance metadata in the same field using some of the remaining bits. Then, using node or edge sampling, we could reconstruct an “AS traceroute” from the packets involved in the DoS attack.

One of the problems the paper encountered was that smart attackers would actually *use* the fragmentation field, intentionally creating fragmented DoS packets, eliminating the ability for sampling. AS routers simply cannot get away from such a problem. My solution to this is to incorporate AS sampling directly into IPv6 packet headers. IPv6 has been well-defined but it is still not actively in use. The IPv6 packet would only need a small 16-bit space to reserve for packet source identification using AS sampling. Such a modification would be a strong step towards deterring Denial of Service activity on the internet today.

V. Conclusion

Unknown attacks are the worst. They can easily go undetected for long periods of time. Capabilities have a lot of potential in that they can completely contain problems that arise from Access Control based systems by moving the metadata from the Objects to the Subjects. However, capabilities are still a long shot by user-friendly standards. Users and programmers are already comfortable with the standard systems that they have always been working with. Detecting Denial of Service sources would send a strong message to attackers out there not to try to mess with systems without being traced. However, adding a new IPv6 field is just as difficult an idea to spread as capabilities are, assuming the new IP version ever gets off of the ground. Knowledge

Based systems will never be enough and detecting behavior will always be faulty, however, to guarantee security we need things like this. Hosts will never get caught off-guard and law enforcement can track down DoS attacks on the internet.

Bibliography

[1] Herve Debar, Marc Dacier and Andreas Wespi. A Revised Taxonomy for Intrusion-Detection Systems. Zurich Research Library, IBM, Switzerland, page numbers, October 1999.

[2] Ravi Sandhu. Access Control: The Neglected Frontier. First Australasian Conference on Information Security and Privacy, Wol-
longong, Australia, June 23-26, 1996.

[3] Christina Warrender, Stephanie Forrest, Barak Pearlmutter. Detecting Intrusions Using System Calls: Alternative Data Models. Dept. Computer Science, page numbers, Albuquerque, NM, October 1998.

[4] Dorothy E. Denning, An Intrusion-Detection Model. SRI International, page numbers, Menlo Park, CA, 1986.

[5] Koral Ilgun, Richard A. Kemmerer, Phillip A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection Approach. IEEE Transactions on Software Engineering, Vol. XX, No. Y, 1995.

[6] J. S. Shapiro, J. M. Smith, D. J. Farber. EROS: A Capability System. *Distributed Systems LaboratoryUniversity of Pennsylvania, Philadelphia, PA, 1997.*

[7] Mark S. Miller, Ka-Ping Yee, Jonathan Shapiro. Capability Myths Demolished. Johns Hopkins University, University of California @ Berkeley, and www.caplet.com, 2003.

[8] Gerald Combs. Ethereal. http://www.ethereal.com.

[9] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, Diego Zamboni. Analysis of a Denial of Service Attack on TCP. COAST Laboratory, Department of Computer Sciences, Purdue University, page numbers, West Lafayette, IN, 1997.

[10] G. Robert Malan, David Watson and Farnam Jahanian. Transport and Application Protocol Scrubbing. Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, 2000.

[11] Controlling High Bandwidth Aggregates in the Network. Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. ICSI Center for Internet Research, AT&T Labs Research, Berkeley, California, 2001.

[12] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Network Support for IP Traceback. IEEE/ACM Transactions On Networking, Vol. 9, No. 3, page numbers, June 2001.