

Stop Hacker Attacks at the OS Level

Firewalls aren't enough to protect your system. You must safeguard the operating system and the applications running on it.

By Dr. Yona Hollander and Romain Agostini



TECHNOLOGIES

Firewalls
Intrusion Detection
Systems
Network-based Intrusion
Detection Systems
Intrusion Prevention
Systems

Despite the benefits of current security solutions, hackers continue to circumvent firewalls to gain access to corporate servers. Once there, they can disrupt critical business functions (e-commerce) and gain illegal access to confidential data. To achieve their goals, these intruders attempt to exploit security holes in both the operating system (OS) and in applications running on top of it. New vulnerabilities are discovered daily and the hacking community is primed to find ways and means to exploit these weak spots. To date, system administrators have relied on vendor patches and kludges to plug the holes, but these fixes are seldom available in a timely fashion. The effort to dispatch them to the whole server infrastructure is time-consuming, labor-intensive, and disruptive. Fixing one problem often creates new ones.

Protecting the operating system and the application layers is fundamental to preventing a hacker from destroying files, defacing Web sites, accessing confidential data, and disrupting or crashing Web applications. In a 2000 survey, 60 percent of organizations indicated they have suffered security breaches in the last two years. Another 2000 survey established that over half the respondents reported between two and ten attacks, and 64 percent of those have suffered Web site vandalism. Yet, no existing solution tackles the problem in an adequate manner. There's clearly a need for more advanced technologies, as the future growth of e-business is largely dependent on

the ability of the security market to keep ahead of the hacker community.

Existing solutions aren't enough

Firewalls are the main perimeter protection tool effectively determining which ports into the corporate network are left open. These open ports provide a conduit for the hacker to bypass the firewall and break into a server machine. A great example is port 80 (HTTP protocol), which Web servers use and, therefore, is always left open. An attacker can send a specifically crafted (yet legitimate) HTTP message to a Web server, passing right through the firewall and exploiting vulnerabilities in the Web server. By exploiting a vulnerability in the Web server, this HTTP message causes a chain of events that ultimately lets the intruder obtain privileged access to the Web server machine. This may seem like a far-fetched scenario, but executable programs that do this are widely available for download off the Internet for those who look for them.

A recent example is the MDAC RDS vulnerability found in Microsoft's IIS Web server. MDAC is a package used to integrate Web and database services. It includes the RDS component that provides remote access to database objects through IIS. Exploiting a vulnerability in RDS, provided that several conditions in the target Web site are met, attackers can use the shell () VBA command with System privileges on the Web server, forcing it to

Dr. Yona Hollander is vice president of strategy for ClickNet Security Technologies, a leading provider of proactive anti-hacking solutions for e-business. Hollander is responsible for identifying security market trends and engaging ClickNet's product development resources to engineer new and innovative anti-hacking solutions.

Romain Agostini is director of security research for ClickNet Security Technologies where he manages the enterceptTM Knowledge Acquisition Team (eKat). ClickNet's eKat researches computer and Internet vulnerabilities and develops signatures for ClickNet's entercept product line.

execute highly privileged system commands on their behalf. The firewall sees the communication packets as a valid HTTP stream and doesn't block the data exchange. But the hacker gains privileged access to the Web server, leading to full control of the machine itself. The attackers have cracked the hard shell that the firewall was supposed to be, and gained access to the data content of your Web server.

Typically, the DNS port is also left open. DNS provides name resolution services and is crucial for providing access to sites. The BIND program (on UNIX) is the most popular DNS server and known to be vulnerable. Potential attackers can gain full control of the machine and use several techniques to redirect communication.

Intrusion detection products are perceived to be the complementary solution to perimeter security. Intrusion detection tools are reactive, monitoring tools that let a security expert identify attack attempts. However, these tools have limited detection capabilities and don't provide on-the-spot attack prevention.

Network-based Intrusion Detection Systems (NIDS) employ sensors that record all communication packets on a network segment. This solution is extremely elegant, as no software is installed on production machines, providing a transparent and easy-to-manage solution. Despite its elegance, there are several inherent problems with the NIDS approach.

1. NIDS aren't blocking and therefore can't prevent attacks in real time. They listen to packets on the wire, but don't block the transfer of the packet. In most cases, the packet reaches its destination and is processed prior to its interpretation by the NIDS. As a result, an attack is often successful before the NIDS identifies it. Even when the NIDS was quick enough to identify the attack, it has still very limited means of reaction. It can alert an operator, or attempt to terminate the communication session. Most often, these means are inadequate for real-time termination of the attack and only provide data for some post-mortem analysis.

2. NIDS have fundamental difficulties in identifying many attacks. In their paper, "Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection," Ptacek and Newsham from Secure Networks, Inc., give some important examples of the limitations of NIDS.

These examples demonstrate several basic facts:

- At the wire level, the information is partial, and it's difficult—if not impossible—to build a high-level picture out of the details. Data and configuration information that can easily be retrieved at the machine level aren't known at the wire level. As a result, the context of the monitored system isn't clear and it's difficult to find a match between an ongoing sequence of events and a predefined attack signature. This translates to poor detection accuracy and high chances for false positives. It's also possible to use many coding tricks to bypass the NIDS' detection capability, even though it's theoretically capable of identifying the attack through a signature.
- The NIDS must interpret a large number of protocols and data formats in order to be effective. This makes the system complex and slow.
- The current NIDS solutions aren't scalable and don't have a built-in means for distributing the load. They also aren't effective when the network is switched or when the communication is encrypted.

As a result of these findings, different vendors quickly reacted, complementing their offerings by adding host-based intrusion detection components, leading to a Host-Based Intrusion Detection System (HIDS). A HIDS is installed on every machine and monitors system logs for suspicious or hostile activity.

The currently available HIDS barely remedies the situation. The typical HIDS isn't integrated into the operating system and relies heavily on the operating system's logging capabilities as its primary information source. However, the logging system wasn't built with intrusion detection in mind, and typically provides incomplete information to the HIDS.

Furthermore, as in the NIDS case, attack detection is delayed and done in virtually all cases after the attack has been completed. HIDS scans logs periodically, creating a time interval in which there's no processing, and in which intrusion attempts will go undetected, even though they've been logged. Also, an HIDS can only correlate events if (1) they've been logged by the operating system or any other application, and (2) if the HIDS contains heuristics to equate these events to an intrusion activity. Consequently, HIDS is mainly used as a post-mortem analysis tool to quantify the damage, and to provide relevant forensics information.

What's missing?

The lessons learned from the experience with Intrusion Detection Systems (IDS) are clear. The essential properties of an effective system to stop hackers are:

- The system should provide prevention, not only detection. For the prevention to be effective, the attack detection must be triggered as early as possible, preferably at the request/message level.
- The system should reside on the machine, not at the network level. For all the reasons mentioned earlier, network intrusion detection at the wire level isn't powerful enough, and host-based intrusion detection is too reactive and spotty in nature. As a conclusion, it becomes pretty much unavoidable to install a security system on each machine if it is to be of practical use.
- The protection system should have up-to-date and complete information about the state, context, and configuration of the machine.
- The information should be clear, precise, and easy to interpret.

Why attacks should be stopped at the OS level

The above set of requirements for an effective system to stop hackers implies that it should provide protection at the operating system level. The prime justification for this claim is the basic architecture of the OS.

Most modern operating systems have a very similar architecture. The OS acts as a funnel through which all service requests are piped. A fundamental design principle is to hide all system resources. The only possible way to access the system resources is provided by predefined OS service routines called System Calls. Any attempt to access the system resources directly is blocked. The System Call interface is clearly defined and allows the activation of the appropriate service routines within the OS kernel.

This fundamental design provides many benefits when trying to build a system to stop hackers at the OS level, including:

1. The OS is at the center. Since the OS acts as a funnel, all system requests use the System Call mechanism. Therefore, to monitor system activities, you only need to keep an eye on System Call execution. Even hackers must use the same interface to accomplish their goals.
2. Real-time detection is possible. Since it's possible to intercept the service requests prior to their execution, it's feasible to terminate the execution of a service before any damage is done.
3. Layered security is possible. Even if hackers gain privileged access rights, their activities can still be monitored and controlled. This is extremely important since both firewalls and Intrusion Detection Systems are useless after the attacker has penetrated the system. Conversely, at the OS level, it's possible to monitor every service request, providing many intervention opportunities.
4. Detection is accurate. The System Call interface is well defined. For a successful execution of a service request, the user must provide the predefined parameters in a clear and concise format. As a result, when the System Call is invoked, it's possible to have all relevant information in the right context to provide accurate detection of attack attempts.

Another motive to stop hackers at the OS level is self-protection. The OS is a prime attack target—after a hacker gains privileged access rights, he actually controls the machine, bypassing many access control and authorization mechanisms. Accordingly, most hackers will first concentrate on gaining privileged access rights by attacking the OS itself.

It's important to note that stopping hackers at the OS level provides means to deal with application security. Many application vulnerabilities let hackers gain privileged access. However, to gain these rights, the application must execute System Calls, which in turn make the solution at the OS level effective.

The intrusion prevention approach

Intrusion Prevention Systems (IPS) are a new technology that overcome existing deficiencies in the IDS approach and stop hackers at the OS level. An Intrusion Prevention System includes an agent on each protected machine. This agent functions as a wrapper around the OS kernel, and is implemented as an extension module to the OS kernel.

System Call interception is the basic mechanism the agent uses to intervene with the OS execution. A typical scenario provides a System Call architecture that accepts several parameters including a System Call number. This number identifies the requested system service and is used to locate the address of the requested System Call code. When executed, the System Call instruction (1) switches to kernel mode; (2) using the System Call number, it retrieves the address of the requested service from a predefined pointer table; and (3) it calls the service. This linking mechanism is elegant and lets you locate the services anywhere in kernel memory, relieving the user from knowing the address of the service requested.

System Call interception takes advantage of this linking mechanism by providing a new set of system service routines. At initialization, the IPS code is first loaded into the kernel as a kernel extension module. Then, when the IPS routine

addresses are known, the appropriate entries in the pointer table are updated.

After initialization, when a System Call instruction executes, the control is transferred to the corresponding IPS routine. The IPS routine compares the incoming parameters against a knowledgebase holding attack signatures. If the call is found to be malicious, the agent can fail its execution. Alternatively, when the request is legitimate, execution proceeds normally.

The Intrusion Prevention System, is transparent and effectively prevents an attack before any damage occurs. Due to the OS architecture, the IPS controls any interaction of the user with the OS. It's activated at the right time just before the system service is executed. When it gains control, it has the appropriate context information as the System Call parameters provide the relevant information regarding the service request. In most cases, it's possible to accurately determine the user identity as well as the module that initiated the call. As a result, the detection of malicious requests is accurate and false positives are rare.

A typical attack scenario

Buffer overflows are a good example to illustrate the effectiveness of the intrusion prevention approach. Buffer overflow vulnerabilities have been the main source of intrusion for many years. A 1999 report by The Hurwitz Group about buffer overflows states:

"By any measure, buffer overflow attacks are an enormous security problem. A recent Hurwitz Group analysis of Computer Emergency Response Team (CERT) advisories revealed that buffer overflow security holes account for more than 50 percent of overall CERT advisories."

As already mentioned, the number one goal of an intruder is to gain privileged (administrator or root) access to the target system. The most probable attack strategy to gain privileged rights is either to trick a vulnerable, privileged program into executing some malicious code, or to plant this malicious code on the machine and find a flaw that allows executing it in privileged mode.

Buffer overflow vulnerabilities provide such an opportunity. For a buffer overflow vulnerability to be successfully exploited, the susceptible program should be privileged. Fortunately—or unfortunately—there are many privileged programs in any operating system. For example, the print utility provides a non-privileged user access to the printer (which is a privileged resource), but shouldn't allow access to any other classified resources. However, existing operating systems don't provide this fine granularity. In all popular operating systems such as Windows and UNIX, a program is either privileged or not.

A buffer overflow vulnerability exists due to bad programming. It happens when an external input is copied into a destination buffer in memory that's too small to hold the input. As a result, the data overflows the destination buffer and overwrites adjacent memory locations. When the destination buffer is located on the machine stack, this vulnerability becomes serious: The machine stack holds both variables (data) and return addresses, and these return addresses can be used to determine what code segments should be executed next.

To exploit the buffer overflow vulnerability, the attackers' goal is to inject a code section into the vulnerable program memory and execute it. The attacker carefully crafts the spe-

cific exploit code, which is a sequence of machine language instructions that include the code to be executed as well as a pointer that serves as the return address and directs the execution into the exploit code.

During the assault, the attacker transfers the exploit code to the vulnerable program. Typically, the hacker executes the vulnerable program and passes the exploit code as an argument. The exploit code can also be transferred through environment variables, file content, communications messages, etc.

After the exploit code has been successfully injected and executed, it can be used to perform any desirable action. In most cases, the hacker will want to spawn a shell. Since a privileged program spawns the shell, the shell will inherit the elevated privileges of the parent. Such shells provide the hacker with unlimited access and ownership of the machine.

A well-known example that demonstrates this technique is the Windows NT buffer overflow in winhlp32.exe. The buffer overflow occurs when winhlp32.exe attempts to read a .CNT file (help contents file) with an overlying, long heading string. The intrusion code loads several missing libraries that finally let it execute the system () function that can run any desired batch file. If the intruder adds a malicious batch file with the appropriate name, the outcome of this attack is that this malicious batch file is executed with elevated privileges.

The attack happens in two stages: The malicious executable runs and modifies a given .CNT file, and the batch file is put somewhere on the disk. At the second stage, an innocent user activates winhlp32.exe using the modified .CNT file. As a result, the batch file is executed. The batch file inherits the privileges of the user running winhlp32.exe. When a privileged user, such as an Administrator, runs winhlp32.exe, the batch file has rights to execute any task to which the Administrator has access.

It's important to note that any exploit code will execute at least one System Call. The reason is simple: If the exploit code doesn't execute any System Call, it won't be able to access any privileged resource or service and attackers won't achieve their goals. The fact that a System Call is invoked is a key point that lets Intrusion Prevention Systems protect against buffer overflow attacks.

How intrusion prevention stops buffer overflow attacks

An Intrusion Prevention System is the most effective solution currently available against buffer overflow attacks. During a buffer overflow attack, the system has multiple points where it's able to intervene, identify, and prevent the attack.

Concentrating on the initiation stage of the attack, and assuming the case where the exploit code is transferred by means of parameter passing, the OS provides a specific System Call to load and execute a given program. The intrusion prevention system intercepts this call and parses the arguments passed to the executable. If one of the arguments includes a long string of binary characters, there's a good chance a buffer overflow attack is in progress.

A similar scenario is when the exploit code is transferred via an environment variable. In this case, the IPS intercepts the same System Call that loads the executable, scans the provided environment variables, and identifies the binary argument.

The above identification techniques are somehow limited since they depend on the means used to transfer the exploit

code to the vulnerable program. It's easy to come up with scenarios where the exploit code is read from a file or sent as part of the message. In these cases, the existence of a binary string isn't always unacceptable and, as a result, the chance for a false positive is much higher.

The other possible opportunity to detect a buffer overflow is when the exploit code is being executed. Since the exploit code invokes at least one System Call, it's guaranteed that the IPS has at least one chance to prevent the attack. This technique is very attractive since it's independent of the mechanism used to pass the exploit code. Furthermore, this technique can prevent unknown buffer overflow attacks.

Benefits of IPS

The experience with intrusion detection systems suggests that effective protection against hacker attacks must be provided on corporate machines and should be tightly coupled with the OS. We've presented the Intrusion prevention approach that employs a System Called-interception to monitor malicious activities. IPS is proactive and can prevent attacks on the spot before any damage happens.

Since IPS is running at the OS level, it benefits from (1) sufficient information at the right context; and (2) the information is given in a clear and well-defined format. This information makes it possible to achieve accurate identification of attacks and lowers the false positive probability significantly. As a result, IPS overcomes many of the inherent problems encountered in existing Intrusion detection products. In addition, the close proximity to the OS lets it harden the protection of the OS itself. This is extremely important since the prime target of hackers is the OS. IPS solutions also provide protection against buffer overflow attacks. This solution is currently the best available means for protecting against this fundamental and widespread problem. **ADVISOR**



ClickNet Security Technologies
2460 Zanker Road
San Jose, CA 95131-1154
Phone: (408) 576-5900 or (800) 599-3200
Fax: (408) 576-5901 or (800) 308-3777
E-mail: sales@clicknet.com