One Step Ahead

Security vulnerabilities abound, but a few simple steps can minimize your risk.

VLAD GORELIK, SANA SECURITY

"When we were children, we used to think that when we were grown-up we would no longer be vulnerable. But to grow up is to accept vulnerability... To be alive is to be vulnerable."

—Madeleine L'Engle

Every day IT departments are involved in an ongoing struggle against hackers trying to break into corporate networks. A break-in can carry a hefty price: loss of valuable information, tarnishing of the corporate image and brand, service interruption, and hundreds of resource hours of recovery time. Unlike other aspects of information technology, security is adversarial—it pits IT departments against hackers.

To explain why one wins in an adversarial situation, the military has a tool called the OODA loop, created by Col. John Boyd, USAF (Ret). As shown in figure 1, OODA is a circular sequence of steps—observe, orient, decide, and act—that describes the decision-action process of each adversary. The idea is that if you can get inside the decision loop of your opponent, you will win the battle. Speed is a key criterion for victory, but other underlying factors also contribute to winning. These factors include quality and completeness of the information gathered during the observe and orient phases, as well as how well the decisions can be carried out. The same concept applies to cybersecurity—those who can make the right decisions faster based on the best information available, and act on them, will win.

Vulnerability management is one of the key aspects of risk management in cybersecurity. To stay inside the attacker's OODA loop, the IT department needs to find and remediate vulnerabilities faster than an attacker can find and exploit them.

One Step Ahead

There are different types of vulnerabilities. One type is a software bug or design flaw that can be exploited to compromise a system. One common vulnerability is the buffer overflow bug, where code does not check for the size of the buffer during a memory copy operation. An attacker can craft a special buffer to insert arbitrary code into a vulnerable process, thus gaining control over a machine or an application.

Other types of vulnerabilities are related to the system, network, or application configuration and are by no means less dangerous. Some common examples of these are default guest accounts, weak passwords, incorrect authorizations, and so on. The only positive aspect of configuration vulnerabilities is that they can often be fixed without third-party patches. Rogue devices are another type of vulnerability, including nonsanctioned Web sites, wireless access points, and nonsanctioned VPNs or outside connections.

HOW BIG IS THE PROBLEM?

The number of software vulnerabilities discovered has significantly spiked in the past two years as a result of boun-

ties offered by security companies and organized crime, as well as better tools for finding vulnerabilities. These tools include fuzz testers, or fuzzers, which can automatically generate random input to stress any border conditions of an application.¹ According to the CERT (Computer Emergency Response Team) Coordination Center at Carnegie Mellon, the number of reported vulnerabilities climbed from 3,780 in 2004 to 5,990 in 2005. The 2005 levels were almost reached in the first three quarters of 2006 (see figure 2).²

All software, whether closed or open source, has vulnerabilities. In 2005 there were 812 Windows operating system vulnerabilities; 2,328 Unix/Linux vulnerabilities; and 2,058 multiple operating system vulnerabilities, as reported by US-CERT.³ Although statistically it seems that open source has more vulnerabilities, it is pretty hard to make a comparison based on counts alone since each vulnerability has a different impact and severity. On the other hand, it is pretty common for open source fixes to be created much quicker than closed source patches. In the first half of 2006 Microsoft Internet Explorer had 38 vulnerabilities and Mozilla Firefox had 47, but the remediation times were nine days and one day on average, respectively.⁴ This time difference helps in closing the OODA-loop time.

VULNERABILITY TIME LINE

Discovery and disclosure have a huge impact on the risks that vulnerabilities pose. The critical points in the life of a vulnerability are:

- When it is discovered
- When and how it is publicly disclosed
- When an exploit for the vulnerability is created
- When a patch/fix is available to remediate the vulnerability
- When a patch/fix is applied to the vulnerable systems

If a black-hat hacker discovers the vulnerability, he or she will typically not publicize it but quietly work on an exploit code. Once an exploit is created, the real danger begins. How the exploit is used depends on the motive of the attacker. It used to be that most attackers were interested in glory—they created worms such as Blaster or Slammer that rapidly impacted thousands of machines and made international news. If the motivation is money, on the other hand, the attacker will likely use the exploit

either to quietly distribute malware across a large number of machines or to hack into high-value targets to steal information. One group of hackers may sell the vulnerability or exploit, and another group will leverage it. Either way, this is the worst-case scenario—the exploit is available prior to the fix. These *zero-day* exploits pop out of the blue and force IT into emergency response and clean-up operations. In recent years several highly publicized zero-

day exploits have occurred on the Windows platform, such as WMF vulnerability (MS 06-001)⁵ and VML vulnerability (MS 06-055)^{6,7} in Internet Explorer.

If a legitimate security researcher finds a vulnerability, he or she can either disclose the vulnerability to the world (i.e., *full disclosure*) or report it to the vendor or group that owns the vulnerable software. The problem with full disclosure is that an attacker can use the information to create an exploit before a fix is available. By disclosing the vulnerability only to the software developer, it will probably not be announced until a patch is available. Unfortunately, not all vendors disclose their vulnerabilities even after they are fixed. This puts the IT departments in danger since they cannot properly evaluate whether a fix should be applied.

Once a fix is available, a new race begins between applying the fix and exploiting the vulnerability. In an open source world, having the fix and the broken code provides an attacker with a detailed

blueprint of how to create an exploit. With the advent of a binary diffing technology,⁸ the attackers can create an exploit from a binary patch with almost the same ease. The time to create an exploit in closed source software has been dramatically reduced from months to days and hours.

For an IT department the clock for the vulnerability management process starts from the time a vulnerability is made public through full disclosure, zero-day exploit, or patch availability. The time from the announcement of the vulnerability to a point where the machine is patched is the *risk window*. One of the goals of vulnerability management is to reduce the risk window or mitigate the risk during that period. (See accompanying sidebar.)

VULNERABILITY MANAGEMENT

Vulnerability management is a process to minimize risk from vulnerabilities. This process is circular like an OODA loop and typically has the following steps:

- Assessment
- Evaluation/Prioritization
- Mitigation
- Remediation

These steps map closely to the OODA decision-action process.

Vulnerabilities are announced by vendors or various vulnerability distribution services. There are many ser-

One Step Ahead

vices—free and paid—that provide alerts on new vulnerabilities, as well as a historic vulnerabilities database.⁹ The standard vulnerability identification is CVE (Common

Vulnerability and Exposure)¹⁰ from the CERT Coordination Center, but many vendors have their own nomenclatures.

On the Receiving End

Eric Allman, Sendmail

A sad truism is that to write code is to create bugs (at least using today's software development technology). The really sad part is that at least some of these are likely to be security bugs. We know how to ameliorate those security bugs (e.g., run your program in a virtual machine), but that does not eliminate them entirely.

The corollary to this is that to distribute a program means that sooner or later you face the possibility of having to manage a security "incident." There is no "correct" way to handle a problem—a lot will depend on hard-to-quantify factors such as how popular the program is, the environments that it runs in, whether it is open or closed source, whether it was discovered internally or externally, etc.

Approach

There are several approaches to dealing with security problems. Here are a few of the more popular (but not necessarily good) ones:

Ignore the problem. This one is very popular, which is surprising given that it doesn't work. The logic is that maybe if you (the software producer) don't publicize the problem, it won't get exploited. This approach is especially popular if the bug is discovered in-house, even more so in closed source software. But are you willing to stake your reputation on it? If the problem is discovered by an outside security researcher, you might be able to get away with claiming you didn't know about it, but if it ever becomes known that you were hiding the problem, you will be publicly eviscerated. This is not generally considered pleasurable.

Silently patch. A more common variant is to silently patch the problem in the next release. This is certainly somewhat better and can sometimes even work if your users are the rare ones who always update when you release a new version. This is seldom the case, however, and almost inevita-

bly some significant set of your users will be running old versions. When the problem does get discovered (and it nearly always does, eventually), your users will often blame you for not telling them that they were exposed. In particular, many customers (quite reasonably) subscribe to the "if it ain't broke, don't fix it" philosophy—to get them to upgrade, you have to give them a good reason.

Patch and announce without details. This approach involves telling the world that there is a security problem, but doing so without divulging any details. Security groups (both outside researchers and those within your customers' organizations), however, understandably like to be able to verify that the problem was appropriately addressed. In general, not announcing at least some details has the same effect as poking at a hornet's nest. This isn't a simple binary decision—for example, few vendors will release an exploit script, even though they developed one in the process of fixing the bug. Announcing a security bug marks the beginning of a footrace between the attackers' ability to exploit and the customers' ability to patch, and you don't want to give the attackers a head start.

Patch with full disclosure. Particularly popular in the open source world (where releasing a patch is tantamount to full disclosure anyway), this involves opening the kimono and exposing everything, including a detailed description of the problem and how the exploit works. It doesn't necessarily mean releasing an exploit script, but sometimes this is unavoidable, particularly if the problem was discovered by one of the more aggressive security groups.

Announce without patch. This isn't normally popular among vendors, but it is sometimes necessary if the problem is already known or will soon be exposed. Such an announcement (hopefully) comes with a workaround. The worst case, of course, is a severe vulnerability with no known workaround that you are forced to reveal. At that point you need to be thinking in terms of damage control.

The goal of the assessment step is to identify where the newly announced vulnerabilities exist in a specific environment. During the evaluation stage fixes are identified and prioritized. The mitigation and remediation stages carry out the changes needed to fix the problems that cause the vulnerability.

Assessment. There are several methods for determining whether you are vulnerable. One is using information

from an inventory audit coupled with known vulnerabilities. It provides a first step to understanding where you might be vulnerable. Symantec's Deep Sight is a good example of this approach. The upside to these types of services is that they filter the massive numbers of vulnerability alerts to those that are applicable only to your environment. The downside here is that your output is only as good as the quality of the data in your inventory.

Due Credit

By the way, if the problem was found externally, it's usually good practice to give credit to the people who found the vulnerability when you make an announcement. Some companies don't like to admit that a problem was found externally, but it is ultimately better to build a good relationship with the security group, and credit is part of that. Most groups will look at the same software over and over again, so you are likely to hear from them more than once.

Timing

Another important consideration, regardless of the approach you choose, is the timing of a security release. Again, there is no correct answer, but there are a few things to consider. Your general rule should be: "Release as soon as possible, but no sooner." If your product is closed source, not a large target, and the exploit was discovered internally, you probably have some breathing room to do the job right, including producing good patch scripts and documentation, coordinating with other affected vendors, etc. On the other hand, if you learned about the exploit because some cracker group just released a fast-spreading worm that uses your code to propagate and gives the attackers complete read/write access to sensitive customer data, you will be on a much tighter schedule, and you may have to cut some corners.

How severe is the exploit? A bug that gives an external user full control of a machine is more critical than one that allows the external user to break into the account of another user who opened an attachment (which that user shouldn't have opened in the first place). Breaking into even a simple non-admin account, however, is generally still enough to turn that machine into a spam zombie, even if it doesn't give access to customer credit-card numbers, so don't be too complacent.

Is the bug discovered internally or externally? If it is discovered externally, then you may not have the option of choosing when the problem is announced. Some security groups will give you a deadline and sometimes will work with you to do an orderly release. None of them will agree to

keep it quiet forever, and if you try to stall, they will generally react negatively. If the group is legitimate (i.e., one that isn't trying to blackmail you), then you can usually negotiate, but only up to a point. Remember, even if you disagree with them, most of those groups are on the right side. Treat them with respect.

Are other vendors affected? If your code is included in other distributions or you have OEMed it to partners, then you owe it to your partners to give them a heads up before you go public, even if they get only a small amount of time (blindsiding your partners is never a good idea).

Is your code open or closed source? Generally speaking, it is easier to find bugs in open source code, and hence the pressure to release quickly may be higher. This isn't a condemnation of open source code. Ross Anderson of Cambridge University found that bugs will get discovered in any case; it's just a matter of when ("Security in Open versus Closed Systems—The Dance of Boltzmann, Coase and Moore," 2002). In other words, there really is no security difference between open and closed source code, but the bugs will generally get fixed sooner in open source code than in closed source code.

Have you tested your patch? As with any bug, the obvious solution isn't necessarily the correct one, and even if there is an active exploit in circulation, you aren't doing your customers any favors by making them install a patch that doesn't protect them, forcing them to do it again. As the saying goes, sometimes when you're up to your ass in alligators, it's hard to remember that you're there to drain the swamp, but do try to do the job right the first time.

How bureaucratic is your company? Unfortunately, sometimes your own company can seem to turn into the enemy. Like it or not, you have to work within the constraints imposed by management. If they are totally clueless, consider showing them this magazine. If that doesn't work, get your résumé in order, and good luck.

ERIC ALLMAN is the cofounder and chief science officer of Sendmail, one of the first open source-based companies.

One Step Ahead

Another process is a vulnerability scan. Tools such as open source Nessus scan the network for known vulnerabilities and identify problem areas. There are typically two types of scans or probes: passive and active. The passive probe simply examines the banner flags on the packets to determine the software and version and correlates it to a vulnerability database. The active probe actually performs an exploit to verify that the machine is vulnerable. Active probes are more accurate but tend to disrupt normal operations.

Penetration testing is usually the most effective but also the most expensive assessment method. A white-hat hacker is hired to try to break into the network or system. There is no substitute for human ingenuity.

Prioritization. In a perfect world it would be possible to fix all vulnerabilities in an instant. In reality there are never enough time and resources to address everything simultaneously. That means that fixing vulnerabilities requires prioritization. The prioritization scheme is a function of the severity of the vulnerability and the context in which it exists. Severity depends on factors such as how easy it is to exploit, whether an exploit already exists, can an exploit be performed remotely, what type of access can an attacker gain from the exploit, how common is the application or system that's vulnerable, and many others. The context in which the vulnerability exists or the risk profile of the machine depends on how accessible it is, what business function it performs, how valuable the data is on this machine, can the machine be used as a jump-off platform for an attacker, etc.

Often machines are prioritized by whether they are inside or outside the network perimeter, making the simple assumption that inside the perimeter is safe. The notion of a secure perimeter around a less secure network is evaporating. Many partner networks connect to corporate LANs, and mobile users connect their laptops to in-

secure networks only to plug them into the corporate LAN a few hours later, bypassing all the security protection. Threats from rogue employees and partners exist inside the perimeter. All of these factors should be considered before using the perimeter as a prioritization criterion.

Mitigation and remediation. Once the priority of applying the fixes is established, it's time to execute them. The only way to truly fix a vulnerability from a software bug is to apply a patch that fixes the broken code. Patching is not without risk, however. Patches are created under great pressure and need testing to make sure they haven't broken functionality of the software or made it incompatible. This testing cycle extends the time before

the patch can be applied. Furthermore, rolling out patches smoothly across potentially hundreds or thousands of machines is no simple matter.

To gain time for applying the patches properly, various mitigation techniques are used to reduce the risk of the vulnerability. One form of mitigation is an IPS (intrusion prevention system), some of which work on the host (HIPS) and some on the

network (NIPS). Although an IPS takes time and effort to set up and tune, once in place it can be used to help block exploits.

IPSeS are not perfect and are susceptible to false positives. Most can run in passive detect mode, as well as block mode. Many IPS deployments run in a minimal block mode to reduce the possibility of blocking a false positive event. One technique is to dynamically put an IPS into blocking mode on vulnerable applications or machines before a patch can be deployed. This provides a good trade-off by applying the IPS blocking to areas with the highest risk and then reducing the blocking once a patch is applied.

At times a system is left vulnerable because it's running a legacy application that is incompatible with a patch—in the real world features and functionality often

win over security concerns. This is another place where IPS-style mitigation is appropriate. Mitigation is also the only resource available in the case of zero-day exploits or full disclosure before a patch is made available by the vendor. Other forms of mitigation disable certain features or reduce access to a part of the network. Most mitigation strategies have some form of negative impact and require a trade-off between the vulnerability risk and the mitigation downside. This trade-off is difficult and ultimately is a business decision rather than a technical one.

Fixing configuration vulnerabilities requires reconfiguring the system and/or application. In some cases a system is misconfigured to allow for a business application to run. Tweaking the different configurations to get things to work properly and retesting the application takes time and effort. This is another place where mitigation might be required.

THE IT DISADVANTAGE

Vulnerability management is all about prevention and risk reduction. All software is vulnerable, and risk needs to be managed on an ongoing basis. Speed and coverage of the vulnerability management process are key to reducing risk—IT departments need to stay inside the hacker's OODA loop. Unfortunately, in cybersecurity the IT departments start with some inherent disadvantages.

First, there are asymmetries in costs:

- Cost of a failed attack for the attacker is negligible compared with the cost of a successful attack for the victim. This means that attackers can keep trying until they succeed or simply get tired and move on.
- Cost of the infrastructure for the attacker is negligible compared with the cost of the infrastructure for defense. Usually, all you need to be an attacker is a computer and Internet access—a lot of the software can be downloaded for free. Even with the new underground market for exploits and malware, an exploit can sell for as little as \$500 and can be used over and over again. Anti-virus and patching software can easily cost \$100 per desktop machine, not including the cost of the time it takes to set up and maintain it. This is especially true when a defender needs to protect multiple targets, whereas the attacker will often be satisfied with breaking into only a small portion of the machines.

Another disadvantage for IT departments is awareness: the attacker knows at least something about the defender, whereas the defender usually knows nothing about the attacker until the attack happens.

A third disadvantage is dependence. The defender often depends on third-party vendors to help with their

defense. These vendors provide the patches needed to remediate the vulnerability or signatures provided by the anti-virus vendors.

These IT disadvantages need to be factored into the process. But no matter how much effort is invested in vulnerability management, there will be times when something or someone will get through—therefore, incident response and contingency planning are always necessary. Q

REFERENCES

1. University of Wisconsin. 2006. Fuzz testing of application reliability; <http://www.cs.wisc.edu/~bart/fuzz/fuzz.html>.
2. CERT/CC Statistics 1988-2006; <http://www.cert.org/stats/>.
3. US-CERT. Cyber Security Bulletin 2005 Summary; <http://www.us-cert.gov/cas/bulletins/SB2005.html#UnixLinux>.
4. Symantec Internet Security Threat Report Trends for January 06-June 06; <http://www.symantec.com/enterprise/threatreport/index.jsp>.
5. Microsoft Security Bulletin MS06-001. 2006. Vulnerability in graphics rendering engine could allow remote code execution; <http://www.microsoft.com/technet/security/bulletin/ms06-001.mspx>.
6. Common Vulnerabilities and Exposures; <http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4868>.
7. Microsoft Security Bulletin MS06-055. 2006. Vulnerability in vector markup language could allow remote code execution; <http://www.microsoft.com/technet/security/bulletin/ms06-055.mspx>.
8. Flake, H. 2004. Structural comparison of executable objects. DIMVA 2004 (Detection of Intrusions and Malware and Vulnerability Assessment).
9. SecurityFocus; <http://www.securityfocus.com>. Secunia; <http://www.secunia.com>.
10. Common Vulnerabilities and Exposures; <http://cve.mitre.org>.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

VLAD GORELIK is the chief technology officer at Sana Security, where he has spent the past several years leading the company's efforts in creating technology to fight malware. He holds multiple patents and patent applications in software technology and computer security.

© 2007 ACM 1542-7730/07/0200 \$5.00