Appendix

The Laws of Security

By Ryan Russell

This book contains a fictional account of a zero day exploit, demonstrating criminal hacking techniques that are used every day to exploit vulnerabilities. While this story is fictional, the dangers are obviously real. As such, we've included this appendix, which discusses how to mitigate attacks, such as the one described in this book. While not a complete reference, these security laws can provide you with a foundation of knowledge to prevent criminal hackers from hacking your network and exploiting your vulnerabilities…

Introduction

One of the shortcuts that security researchers use in discovering vulnerabilities is a mental list of observable behaviors that tells them something about the security of the system they are examining. If they can observe a particular behavior, it is a good indication that the system has a trait that they would consider to be insecure, even before they have a chance to perform detailed tests.

We call our list the *Laws of Security*. These laws are guidelines that you can use to keep an eye out for security problems while reviewing or designing a system. The system in this case might be a single software program, or it could be an entire network of computers, including firewalls, filtering gateways, and virus scanners. Whether defending or attacking such a system, it is important to understand where the weak points are.

The Laws of Security will identify the weak points and allow you to focus your research on the most easily attackable areas. This Appendix concerns itself with familiarizing you with these laws.

Knowing the Laws of Security

The laws of security in our list include:

- Client-side security doesn't work.
- You cannot securely exchange encryption keys without a shared piece of information.
- Malicious code cannot be 100 percent protected against.
- Any malicious code can be completely morphed to bypass signature detection.
- Firewalls cannot protect you 100 percent from attack.
- Any intrusion detection system (IDS) can be evaded.
- Secret cryptographic algorithms are not secure.
- If a key isn't required, you do not have encryption—you have encoding.

- Passwords cannot be securely stored on the client unless there is another password to protect them.
- In order for a system to begin to be considered secure, it must undergo an independent security audit.
- Security through obscurity does not work.

There are a number of different ways to look at security laws. In this Appendix, we've decided to focus on *theory*, or laws that are a bit closer to a mathematical rule. (At least, as close as we can get to that type of rule. Subjects as complex as these don't lend themselves to formal proofs.) There's another way to build a list of laws: we could make a list of not what is *possible*, but what is *practical*. Naturally, there would be some overlap—if it's not possible, it's also not practical. Scott Culp, Microsoft's Security Response Center Manager, produced a top-ten list of laws from the point of view of his job and his customers. He calls these "The Ten Immutable Laws of Security." They are:

- Law #1: If a bad guy can persuade you to run his program on your computer, it's not your computer anymore.
- Law #2: If a bad guy can alter the operating system on your computer, it's not your computer anymore.
- Law #3: If a bad guy has unrestricted physical access to your computer, it's not your computer anymore.
- Law #4: If you allow a bad guy to upload programs to your Web site, it's not your Web site any more.
- Law #5: Weak passwords trump strong security.
- Law #6: A machine is only as secure as the administrator is trustworthy.
- Law #7: Encrypted data is only as secure as the decryption key.
- Law #8: An out-of-date virus scanner is only marginally better than no virus scanner at all.
- Law #9: Absolute anonymity isn't practical, in real life or on the Web.

- Law #10: Technology is not a panacea.

The full list (with explanations for what each rule means) can be found at www.microsoft.com/technet/columns/security/10imlaws.asp. This list is presented to illustrate another way of looking at the topic, from a defender's point of view. For the most part, you will find that these laws are the other side of the coin for the ones we will explore.

Before we can work with the laws to discover potential problems, we need to have a working definition of what the laws are. In the following sections, we'll look at the laws and what they mean to us in our efforts to secure our networks and systems.

Client-Side Security Doesn't Work

In the first of our laws, we need to define a couple of concepts in regard to security. What, exactly, are we talking about when we begin to discuss "client-side?" If we were in a network (client-server) environment, we would define the client as the machine initiating a request for service and connection, and the server as the machine waiting for the request for service or connection or the machine able to provide the service. The term "client-side" in the network is used to refer to the computer that represents the client end, that over which the user (or the attacker) has control. The difference in usage in our law is that we call it client-side even if no network or server is involved. Thus, we refer to "client-side" security even when we're talking about just one computer with a piece of software on a floppy disk. The main distinction in this definition is the idea that users (or attackers) have control over their own computers and can do what they like with them.

Now that we have defined what "client-side" is, what is "client-side security?" Client-side security is some sort of security mechanism that is being enforced *solely on the client*. This may be the case even when a server is involved, as in a traditional client-server arrangement. Alternately, it may be a piece of software running on your computer that tries to prevent you from doing something in particular.

The basic problem with client-side security is that the person sitting physically in front of the client has absolute control over it. Scott Culp's Law #3 illustrates

this in a more simplistic fashion: *If a bad guy has unrestricted physical access to your computer, it's not your computer anymore.* The subtleties of this may take some contemplation to fully grasp. You cannot design a client-side security mechanism that users cannot eventually defeat, should they choose to do so. At best, you can make it challenging or difficult to defeat the mechanism. The problem is that because most software and hardware is mass-produced, one dedicated person who figures it out can generally tell everyone else in the world, and often will do so. Consider a software package that tries to limit its use in some way. What tools does an attacker have at his or her disposal? He or she can make use of debuggers, disassemblers, hex editors, operating system modification, and monitoring systems, not to mention unlimited copies of the software.

What if the software detects that it has been modified? Remove the portion that detects modification. What if the software hides information somewhere on the computer? The monitoring mechanisms will ferret that out immediately. Is there such a thing as tamper-proof hardware? No. If an attacker can spend unlimited time and resources attacking your hardware package, any tamper proofing will eventually give way. This is especially true of mass-produced items. We can, therefore, generally say that client-side security doesn't work.

You Cannot Securely Exchange Encryption Keys without a Shared Piece of Information

Although this law may seem obvious if you have worked with encryption, it presents a unique challenge in the protection of our identities, data, and information exchange procedures. There is a basic problem with trying to set up encrypted communications: exchanging session keys securely. These keys are exchanged between the client and server machines prior to the exchange of data, and are essential to the process

To illustrate this, let's look at setting up an encrypted connection across the Internet. Your computer is running the nifty new CryptoX product, and so is the computer you're supposed to connect to. You have the IP

address of the other computer. You type it in and hit **Connect**. The software informs you that it has connected, exchanged keys, and now you're communicating securely using 1024-bit encryption. Should you trust it? Unless there has been some significant crypto infrastructure set up behind it (and we'll explain what that means later in this Appendix), you shouldn't. It's not impossible, and not necessarily even difficult, to hijack IP connections.

The problem here is how do you *know* what computer you exchanged keys with? It might have been the computer you wanted. It might have been an attacker who was waiting for you to make the attempt, and who pretended to be the IP address you were trying to reach. The only way you could tell for certain would be if both computers had a piece of information that could be used to verify the identity of the other end. How do we accomplish this? A couple of methods come to mind. First, we could use the public keys available through certification authorities that are made available by Web browser providers. Second, we could use Secure Sockets Layer (SSL) authentication, or a shared secret key. All of these, of course, are shared pieces of information required to verify the sender of the information.

This boils down to a question of key management, and we'll examine some questions about the process. How do the keys get to where they are needed? Does the key distribution path provide a path for an attacker waiting to launch a man-in-the-middle (MITM) attack? How much would that cost in terms of resources in relation to what the information is worth? Is a trusted person helping with the key exchange? Can the trusted person be attacked? What methods are used to exchange the keys, and are they vulnerable?

Let's look at a couple of ways that keys are distributed and exchanged. When encryption keys are exchanged, some bit of information is required to make sure they are being exchanged with the right party and not falling victim to a MITM attack. Providing proof of this is difficult, since it's tantamount to proving the null hypothesis, meaning in this case that we'd probably have to show every possible key exchange protocol that could ever be invented, and then prove that they are all individually vulnerable to MITM attacks.

As with many attacks, it may be most effective to rely on the fact that people don't typically follow good security advice, or the fact that the encryption end points are usually weaker than the encryption itself.

Let's look at a bit of documentation on how to exchange public keys to give us a view of one way that the key exchanges are handled: www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt4/scencryp.htm#xtocid211509.

This is a document from Cisco Systems, Inc. that describes, among other things, how to exchange Digital Signature Standard (DSS) keys. DSS is a public/private key standard that Cisco uses for peer router authentication. Public/private key crypto is usually considered too slow for real-time encryption, so it's used to exchange symmetric session keys (such as DES or 3DES keys). DES is the Data Encryption Standard, the U.S. government standard encryption algorithm, adopted in the 1970s. 3DES is a stronger version of it that links together three separate DES operations, for double or triple strength, depending on how it's done. In order for all of this to work, each router has to have the right public key for the other router. If a MITM attack is taking place and the attacker is able to fool each router into accepting one of his public keys instead, then he knows all the session keys and can monitor any of the traffic.

Cisco recognizes this need, and goes so far as to say that you "must verbally verify" the public keys. Their document outlines a scenario in which there are two router administrators, each with a secure link to the router (perhaps a terminal physically attached to the console), who are on the phone with each other. During the process of key exchange, they are to read the key they've received to the other admin. The security in this scenario comes from the assumptions that the two administrators recognize each other's voices, and that it's very difficult to fake someone else's voice.

If the administrators know each other well, and each can ask questions the other can answer, and they're both logged on to the consoles of the router, and no one has compromised the routers, then this is secure, unless there is a flaw in the crypto.

We're not going to attempt to teach you how to mimic someone else's voice, nor are we going to cover taking over phone company switches to reroute calls for administrators who don't know each other. Rather, we'll

attack the assumption that there are two administrators and that a secure configuration mechanism is used.

One would suspect that, contrary to Cisco's documentation, most Cisco router key exchanges are done by one administrator using two Telnet windows. If this is the case and the attacker is able to play man-in-the-middle and hijack the Telnet windows and key exchange, then he can subvert the encrypted
communications.

Finally, let's cover the endpoints. Security is no stronger than the weakest links. If the routers in our example can be broken into and the private keys recovered, then none of the MITM attacking is necessary. At present, it appears that Cisco does a decent job of protecting the private keys; they cannot be viewed normally by even legitimate administrators. They are, however, stored in memory. Someone who wanted to physically disassemble the router and use a circuit probe of some sort could easily recover the private key. Also, while there hasn't been any public research into buffer overflows and the like in Cisco's IOS, I'm sure there will be someday. A couple of past attacks have certainly indicated that such buffer overflows exist.

Another way to handle the exchange is through the use of SSL and your browser. In the normal exchange of information, if you weren't asked for any information, then the crypto must be broken. How, then, does SSL work? When you go to a "secure" Web page, you don't have to provide anything. Does that mean SSL is a scam? No—a piece of information has indeed been shared: the root certificate authority's public key. Whenever you download browser software, it comes with several certificates already embedded in the installer. These certificates constitute the bit of information required to makes things "secure." Yes, there was an opportunity for a MITM attack when you downloaded the file. If someone were to muck with the file while it was on the server you downloaded it from or while it was in transit to your computer, all your SSL traffic could theoretically be compromised.

SSL is particularly interesting, as it's one of the best implementations of mass-market crypto as far as handling keys and such. Of course, it is not

without its problems. If you're interested in the technical details of how SSL works, check here: www.rsasecurity.com/standards/ssl/index.html.

Malicious Code Cannot Be 100 Percent Protected against

During the last couple of years, we have seen more and more attacks using weaknesses in operating systems and application code to gain entrance to our systems. Recently, we've seen a number of programs that were quickly modified and redeployed on the Internet and have resulted in widespread disruption of service and loss of data. Why is this? It is because we can't protect 100 percent against malicious code when it changes as rapidly as it does now. We'll take a look at some examples of this in the following section and discuss the anti-virus protection process as an example.

If, like most people, you run a Windows-based operating system (and perhaps even if you have something else), you run anti-virus software. Perhaps you're even diligent about keeping your virus definitions up to date. Are you completely protected against viruses? Of course not.

Let's examine what viruses and Trojans are, and how they find their way onto your computer. Viruses and Trojans are simply programs, each of which has a particular characteristic. Viruses replicate and require other programs to attach themselves to. Trojans pretend to have a different function than the one they actually have. Basically, they are programs that the programmer designed to do something you generally would not want to have happen if you were aware of their function. These programs usually get onto your computer through some sort of trickery. They pretend to be something else, they're attached to a program you wanted, or they arrive on media you inserted without knowing it was infected. They can also be placed by a remote attacker who has already compromised your security.

How does anti-virus software work? Before program execution can take place, the anti-virus software will scan the program or media for "bad things," which usually consist of viruses, Trojans, and even a few potential hacker tools. Keep in mind, though, that your anti-virus software vendor is the sole determiner of what to check for, unless you take the time to develop your own signature files. Signature files are the meat of most anti-

virus programs. They usually consist of pieces of code or binary data that are (you hope) unique to a particular virus or Trojan. Therefore, if you get a virus that does not appear in the database, your anti-virus software cannot help you.

So why is the process so slow? In order to produce a signature file, an anti-virus vendor has to get a copy of the virus or Trojan, analyze it, produce a signature, update the signature file (and sometimes the anti-virus program too) and publish the update. Finally, the end user has to retrieve and apply the update. As you might imagine, there can be some significant delays in getting new virus information to end users, and until they get it they are vulnerable.

You cannot blindly run any program or download any attachment simply because you run anti-virus software. Not so long ago, anti-virus software could usually be relied upon, because viruses propagated so slowly, relying on people to move them about via diskettes or shared programs. Now, since so many computers connect to the Internet, that connectivity has become a very attractive carrier for viruses. They spread via Web pages, e-mail and downloads. Chances are much greater now that you will see a new virus before your anti-virus software vendor does. And don't forget that a custom virus or Trojan may be written specifically to target you at any time. Under those circumstances, your anti-virus software will never save you.

I'd like to tell my favorite "virus variant" story. In April 2000, we saw the introduction of the "I Love You" virus via the Internet. This was another of the virus worms running in conjunction with Microsoft's Outlook e-mail program, and had far greater impact because it sent itself to all of the e-mail recipients in the address book rather than just the first fifty, as did the earlier "Melissa" virus. However, despite the efforts of anti-virus vendors and others to contain the virus, it spread rapidly and spawned a number of copycat viruses in the short time after it was introduced. Why couldn't it be contained more quickly? In the case of a number of my clients, it was because there were far too many employees who couldn't resist finding out *who* loved them so much! Containment is not always the province of your security or implementations of protective software.

Trojans and viruses actually *could* be protected against completely by users modifying their behavior. They probably wouldn't get much done with a computer, though. They'd have to install only software obtained directly from a trusted vendor (however one would go about determining that. There have been several instances of commercial products shipping with viruses on the media). They'd probably have to forgo the use of a network and never exchange information with anyone else. And, of course, the computer would have to be physically secure.

Any Malicious Code Can Be Completely Morphed to Bypass Signature Detection

This law is fairly new to our discussions of security, and it has become much more prevalent over the past year. It is a new truth, since the attackers now have the ability to change the existing virus/Trojan/remote control application nearly as soon as it is released in the wild. This leads to the discussion of the new problem—variants. If we continue the discussion with the anti-virus example, we'll find that if there is even a slight change in the virus code, there's a chance that the anti-virus software won't be able to spot it any longer. These problems used to be much less troublesome. Sure, someone had to get infected first, and their systems were down, but chances were good it wouldn't be you. By the time it made its way around to you, your anti-virus vendor had a copy to play with, and you'd updated your files.

This is no longer the case. The most recent set of viruses propagates much, much more quickly. Many of them use e-mail to ship themselves between users. Some even pretend to be you, and use a crude form of social engineering to trick your friends into running them. This year, we have seen the evidence of this over and over as the various versions of the Code Red virus were propagated throughout the world. As you recall, the original version was time and date functional, with a programmed attack at a U.S. government agency's Web site. It was modified successfully by a number of different individuals, and led to a proliferation of attacks that

took some time to overcome. Why was this so successful? The possibilities for change are endless, and the methods numerous. For instance, you can modify the original code to create a new code signature, compress the file, encrypt the file, protect it with a password, or otherwise modify it to help escape detection. This allows you to move past the virus scanners, firewalls, and IDS systems, because it is a new signature that is not yet recognized as a threat.

Tools & Traps…

Want to Check that Firewall?

There are an incredible number of freeware tools available to you for beginning your checks of vulnerability. Basic tools, of course, include the basic Transmission Control Protocol/Internet Protocol (TCP/IP) tools included with the protocol: ping, tracert, pathping, Telnet, and nslookup can all give you a quick look at vulnerabilities. Along with these, I have a couple of favorites that allow for quick probes and checks of information about various IP addresses:

- SuperScan, from Foundstone Corporation: www.foundstone.com/knowledge/free_tools.html (click on SCANNER).
- Sam Spade, from SamSpade.org: www.samspade.org.

These two tools, among many other very functional tools, will allow you to at least see some of the vulnerabilities that may exist where you are.

Firewalls Cannot Protect You 100 Percent from Attack

Firewalls can protect a network from certain types of attacks, and they provide some useful logging. However, much like anti-virus software, firewalls will never provide 100 percent protection. In fact, they often provide much less than that.

First of all, even if a firewall were 100 percent effective at stopping all attacks that tried to pass through it, one has to realize that not all avenues of attack go through the firewall. Malicious employees, physical security, modems, and infected floppies are all still threats, just to name a few. For purposes of this discussion, we'll leave threats that don't pass through the firewall alone.

Firewalls are devices and/or software designed to selectively separate two or more networks. They are designed to permit some types of traffic while denying others. What they permit or deny is usually under the control of the person who manages the firewall. What is permitted or denied should reflect a written security policy that exists somewhere within the organization.

As long as something is allowed through, there is potential for attack. For example, most firewalls permit some sort of Web access, either from the inside out or to Web servers being protected by the firewall. The simplest of these is port filtering, which can be done by a router with access lists. A simple and basic filter for Internet Control Message Protocol (ICMP) traffic blocking it at the outside interface will stop responses from your system to another when an outsider pings your interface. If you want to see this condition, ping or use tracert on www.microsoft.com. You'll time out on the connection. Is Microsoft down? Hardly—they just block ICMP traffic, among other things, in their defense setup. There are a few levels of protection a firewall *can* give for Web access. Simply configure the router to allow inside hosts to reach any machine on the Internet at TCP port 80, and any machine on the Internet to send replies from port 80 to any inside machine. A more careful firewall may actually understand the Hypertext Transfer Protocol (HTTP), perhaps only allowing legal HTTP commands. It may be able to compare the site being visited against a list of not-allowed sites. It might be able to hand over any files being downloaded to a virus-scanning program to check.

Let's look at the most paranoid example of an HTTP firewall. You'll be the firewall administrator. You've configured the firewall to allow only legal HTTP commands. You're allowing your users to visit a list of only 20 approved sites. You've configured your firewall to strip out Java, JavaScript,

and ActiveX. You've configured the firewall to allow only retrieving HTML, .gif, and .jpg files.

Can your users sitting behind your firewall still get into trouble? Of course they can. I'll be the evil hacker (or perhaps the security-ignorant Webmaster) trying to get my software through your firewall. How do I get around the fact that you only allow certain file types? I put up a Web page that tells your users to right-click on a .jpg to download it and then rename it to evil.exe once it's on their hard drive. How do I get past the anti-virus software? Instead of telling your users to rename the file to .exe, I tell them to rename it to .zip, and unzip it using the password "hacker." Your anti-virus software will never be able to check my password-protected zip file. But that's okay, right? You won't let your users get to my site anyway. No problem. All I have to do is break into one of your approved sites. However, instead of the usual obvious defacement, I leave it as is, with the small addition of a little JavaScript. By the time anyone notices that it has had a subtle change, I'll be in.

Won't the firewall vendors fix these problems? Possibly, but there will be others. The hackers and firewall vendors are playing a never-ending game of catch-up. Since the firewall vendors have to wait for the hackers to produce a new attack before they can fix it, they will always be behind.

On various firewall mailing lists, there have been many philosophical debates about exactly which parts of a network security perimeter comprise "the firewall," but those discussions are not of use for our immediate purposes. For our purposes, firewalls are the commercial products sold as firewalls, various pieces of software that claim to do network filtering, filtering routers, and so on. Basically, our concern is *how do we get our information past a firewall?*

It turns out that there is plenty of opportunity to get attacks past firewalls. Ideally, firewalls would implement a security policy perfectly. In reality, someone has to create the firewall, so they are far from perfect. One of the major problems with firewalls is that firewall administrators can't very easily limit traffic to exactly the type they would like. For example, the policy may state that Web access (HTTP) is okay, but RealAudio use is not. The firewall admin should just shut off the ports for RealAudio, right? Problem is, the folks who wrote RealAudio are aware that this might

happen, so they give the user the option to pull down RealAudio files via HTTP. In fact, unless you configure it away, most versions of RealAudio will go through several checks to see how they can access RealAudio content from a Web site, and it will automatically select HTTP if it needs to do so. The real problem here is that any protocol can be tunneled over any other one, as long as timing is not critical (that is, if tunneling won't make it run too slowly). RealAudio does buffering to deal with the timing problem.

The designers of various Internet "toys" are keenly aware of which protocols are typically allowed and which aren't. Many programs are designed to use HTTP as either a primary or backup transport to get information through.

There are probably many ways to attack a company with a firewall without even touching the firewall. These include modems, diskettes, bribery, breaking and entering, and so on. For the moment, we'll focus on attacks that must traverse the firewall.

Social Engineering

One of the first and most obvious ways to traverse a firewall is trickery. E-mail has become a very popular mechanism for attempting to trick people into doing stupid things; the "Melissa" and "I Love You" viruses are prime examples. Other examples may include programs designed to exhibit malicious behavior when they are run (Trojans) or legitimate programs that have been "infected" or wrapped in some way (Trojans/viruses). As with most mass-mail campaigns, a low response rate is enough to be successful. This could be especially damaging if it were a custom program, so that the anti-virus programs would have no chance to catch it. For information about what can be done with a virus or Trojan.

Attacking Exposed Servers

Another way to get past firewalls is to attack exposed. Many firewalls include a demilitarized zone (DMZ) where various Web servers, mail servers and so on are placed. There is some debate as to whether a classic DMZ is a network completely outside the firewall (and therefore not pro-

tected by the firewall) or whether it's some in-between network. Currently in most cases, Web servers and the like are on a third interface of the firewall that protects them from the outside, allowing the inside not to trust them either and not to let them in.

The problem for firewall admins is that firewalls aren't all that intelligent. They can do filtering, they can require authentication, and they can do logging, but they can't really tell a good allowed request from a bad allowed request. For example, I know of no firewall that can tell a legitimate request for a Web page from an attack on a Common Gateway Interface (CGI) script. Sure, some firewalls can be programmed to look for certain CGI scripts being attempted (phf, for example), but if you've got a CGI script you *want* people to use, the firewall isn't going to able to tell those people apart from the attacker who has found a hole in it. Much of the same goes for Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and many other commonly offered services. They are all attackable.

For the sake of discussion, let's say that you've found a way into a server on the DMZ. You've gained root or administrator access on that box. That doesn't get you inside, does it? Not directly, no. Recall that our definition of DMZ included the concept that DMZ machines can't get to the inside. Well, that's usually not strictly true. Very few organizations are willing to administer their servers or add new content by going to the console of the machine. For an FTP server, for example, would they be willing to let the world access the FTP ports, but not themselves? For administration purposes, most traffic will be initiated from the inside to the DMZ. Most firewalls have the ability to act as diodes, allowing traffic to be initiated from one side but not from the other. That type of traffic would be difficult but not impossible to exploit. The main problem is that you have to wait for something to happen. If you catch an FTP transfer starting, or the admin opening an X window back inside, you may have an opportunity.

More likely, you'll want to look for allowed ports. Many sites include services that require DMZ machines to be able to initiate contact back to the inside machine. This includes mail (mail has to be delivered inside), database lookups (for e-commerce Web sites, for example), and possibly

reporting mechanisms (perhaps syslog). Those are more helpful because you get to determine when the attempt is made. Let's look at a few cases:

Suppose you were able to successfully break into the DMZ mail server via some hole in the mail server daemon. Chances are good that you'll be able to talk to an internal mail server from the DMZ mail server. Chances are also good that the inside mail server is running the same mail daemon you just broke into, or even something less well protected (after all, it's an inside machine that isn't exposed to the Internet, right?)

Attacking the Firewall Directly

You may find in a few cases that the firewall itself can be compromised. This may be true for both homegrown firewalls (which require a certain amount of expertise on the part of the firewall admin) and commercial firewalls (which can sometimes give a false sense of security, as they need a certain amount of expertise too, but some people assume that's not the case). In other cases, a consultant may have done a good job of setting up the firewall, but now no one is left who knows how to maintain it. New attacks get published all the time, and if people aren't paying attention to the sources that publish this stuff, they won't know to apply the patches.

The method used to attack a firewall is highly dependent on the exact type of the firewall. Probably the best sources of information on firewall vulnerabilities are the various security mailing lists. A particularly malicious attacker would do as much research about a firewall to be attacked as possible, and then lie in wait for some vulnerability to be posted.

Client-Side Holes

One of the best ways to get past firewalls is client-side holes. Aside from Web browser vulnerabilities, other programs with likely holes include AOL Instant Messenger, MSN Chat, ICQ, IRC clients, and even Telnet and ftp clients. Exploiting these holes can require some research, patience, and a little luck. You'll have to find a user in the organization you want to attack that appears to be running one of these programs, but many of the chat programs include a mechanism for finding people, and it's not uncommon for people to post their ICQ number on their homepage. You could do a

search for victim.com and ICQ. Then you could wait until business hours when you presume the person will be at work, and execute your exploit using the ICQ number. If it's a serious hole, then you now probably have code running behind the firewall that can do as you like.

Any IDS Can Be Evaded

And you ask, "What the heck is an IDS?" IDS stands for *intrusion detection system*. At the time of this writing, there are hundreds of vendors providing combined hardware and software products for intrusion detection, either in combination with firewall and virus protection products or as freestanding systems. IDSs have a job that is slightly different from that of firewalls. Firewalls are designed to stop bad traffic. IDSs are designed to spot bad traffic, but not necessarily to stop it (though a number of IDSs will cooperate with a firewall to stop the traffic, too). These IDSs can spot suspicious traffic through a number of mechanisms. One is to match it against known bad patterns, much like the signature database of an anti-virus program. Another is to check for compliance against written standards and flag deviations. Still another is to profile normal traffic and flag traffic that varies from the statistical norm. Because they are constantly monitoring the network, IDSs help to detect attacks and abnormal conditions both internally and externally in the network, and provide another level of security from inside attack.

As with firewalls and client-side security methods, IDSs can be evaded and worked around. One of the reasons that this is true is because we still have users working hands-on on machines within our network, and as we saw with client-side security, this makes the system vulnerable. Another cause in the case of firewalls and IDS systems is that although they are relatively tight when first installed, the maintenance and care of the systems deteriorates with time, and vigilance declines. This leads to many misconfigured and improperly maintained systems, which allows the evasion to occur.

The problem with IDSs for attackers is that they don't know when one is present. Unlike firewalls, which are fairly obvious when you hit them, IDSs can be completely passive and therefore not directly detectable. They

can spot suspicious activity and alert the security admin for the site being attacked, unbeknownst to the attacker. This may result in greater risk of prosecution for the attacker. Consider getting an IDS. Free ones are starting to become available and viable, allowing you to experiment with the various methods of detection that are offered by the IDS developers. Make sure you audit your logs, because no system will ever achieve the same level of insight as a well-informed person. Make absolutely sure that you keep up-to-date on new patches and vulnerabilities. Subscribe to the various mailing lists and read them. From the attack standpoint, remember that the attacker can get the same information that you have. This allows the attacker to find out what the various IDS systems detect and, more importantly, *how* the detection occurs. Variations of the attack code can then be created that are not detectable by the original IDS flags or settings.

In recent months, IDSs have been key in collecting information about new attacks. This is problematic for attackers, because the more quickly their attack is known and published, the less well it will work as it's patched away. In effect, any new research that an attacker has done will be valuable for a shorter period of time. I believe that in a few years, an IDS system will be standard equipment for every organization's Internet connections, much as firewalls are now.

Secret Cryptographic Algorithms Are Not Secure

This particular "law" is not, strictly speaking, a law. It's theoretically possible that a privately, secretly developed cryptographic algorithm *could* be secure. It turns out, however, that it just doesn't happen that way. It takes lots of public review and lots of really good cryptographers trying to break an algorithm (and failing) before it can begin to be considered secure.

Bruce Schneier has often stated that anyone can produce a cryptographic algorithm without being able to break it. Programmers and writers know this as well. Programmers cannot effectively beta-test their own software, just as writers cannot effectively proofread their own writing. Put another way, to produce a secure algorithm, a cryptographer must know all possible attacks and be able to recognize when they apply to his or her

algorithm. This includes currently known attacks as well as those that may be made public in the future. Clearly no cryptographer can predict the future, but some of them have the ability to produce algorithms that are resistant to new things because they are able to anticipate or guess some possible future attacks.

This has been demonstrated many times in the past. A cryptographer, or someone who thinks he or she is one, produces a new algorithm. It looks fine to this person, who can't see any problem. The "cryptographer" may do one of several things: use it privately, publish the details, or produce a commercial product. With very few exceptions, if it's published, it gets broken, and often quickly. What about the other two scenarios? If the algorithm isn't secure when it's published, it isn't secure at any time. What does that do to the author's private security or to the security of his customers?

Why do almost all new algorithms fail? One answer is that good crypto is hard. Another is the lack of adequate review. For all the decent cryptographers who can break someone else's algorithm, there are many more people who would like to try writing one. Crypto authors need lots of practice to learn to write good crypto. This means they need to have their new algorithms broken over and over again, so they can learn from the mistakes. If they can't find people to break their crypto, the process gets harder. Even worse, some authors may take the fact that no one broke their algorithm (probably due to lack of time or interest) to mean that it must be secure!

For an example of this future thinking, let's look at DES. In 1990, Eli Biham and Adi Shamir, two world-famous cryptographers, "discovered" what they called differential cryptanalysis. This was some time after DES had been produced and made standard. Naturally, they tried their new technique on DES. They were able to make an improvement over a simple brute-force attack, but there was no devastating reduction in the amount of time it took to crack DES. It turns out that the structure of the s-boxes in DES was nearly ideal for defending against differential cryptanalysis. It seems that someone who worked on the DES design knew of, or had suspicions about, differential cryptanalysis.

Very few cryptographers are able to produce algorithms of this quality. They are also the ones who usually are able to break the good algorithms. I've heard that a few cryptographers advocate breaking other people's algorithms as a way to learn how to write good ones. These world-class cryptographers produce algorithms that get broken, so they put their work out into the cryptographic world for peer review. Even then, it often takes time for the algorithms to get the proper review. Some new algorithms use innovative methods to perform their work. Those types may require innovative attack techniques, which may take time to develop. In addition, most of these cryptographers are in high demand and are quite busy, so they don't have time to review every algorithm that gets published. In some cases, an algorithm would have to appear to be becoming popular in order to justify the time spent looking at it. All of these steps take time—sometimes years. Therefore, even the best cryptographers will sometimes recommend that you not trust their own new algorithms until they've been around for a long time. Even the world's best cryptographers produce breakable crypto from time to time.

The U.S. government has now decided to replace DES with a new standard cryptographic algorithm. This new one is to be called Advanced Encryption Standard (AES), and the NIST (National Institute of Standards and Technology) has selected Rijndael as the proposed AES algorithm. Most of the world's top cryptographers submitted work for consideration during a several-day conference. A few of the algorithms were broken during the conference by the other cryptographers.

We can't teach you how to break real crypto. That's okay, though. We've still got some crypto fun for you. There are lots of people out there who think they are good cryptographers and are willing to sell products based on that belief. In other cases, developers may realize that they can't use any real cryptography because of the lack of a separate key, so they may opt for something simple to make it less obvious what they are doing. In those cases, the crypto will be much easier to break

Again, the point of this law is not to perform an action based on it, but rather to develop suspicion. You should use this law to evaluate the quality of a product that contains crypto. The obvious solution here is to use well-established crypto algorithms. This includes checking as much as possible

that the algorithms are used intelligently. For example, what good does 3DES do you if you're using only a seven-character password? Most passwords that people choose are only worth a few bits of randomness per letter. Seven characters, then, is much less than 56 bits.

If a Key Is Not Required, You Do Not Have Encryption—You Have Encoding

This one is universal—no exceptions. Just be certain that you know whether or not there is a key and how well it's managed. As Scott Culp mentions in his law #7, "*Encrypted data is only as secure as the decryption key.*"

The key in encryption is used to provide variance when everyone is using the same small set of algorithms. Creating good crypto algorithms is hard, which is why only a handful of them are used for many different things. New crypto algorithms aren't often needed, as the ones we have now can be used in a number of different ways (message signing, block encrypting, and so on). If the best-known (and foreseeable) attack on an algorithm is brute force, and brute force will take sufficiently long, there is not much reason to change. New algorithms should be suspect, as we mentioned previously.

In the early history of cryptography, most schemes depended on the communicating parties using the same system to scramble their messages to each other. There was usually no key or pass-phrase of any sort. The two parties would agree on a scheme, such as moving each letter up the alphabet by three letters, and they would send their messages.

Later, more complicated systems were put into use that depended on a word or phrase to set the mechanism to begin with, and then the message would be run through. This allowed for the system to be known about and used by multiple parties, and they could still have some degree of security if they all used different phrases.

These two types highlight the conceptual difference between what encoding and encrypting are. Encoding uses no key, and if the parties involved want their encoded communications to be secret, then their encoding scheme must be secret. Encrypting uses a key (or keys) of some

sort that both parties must know. The algorithm can be known, but if an attacker doesn't have the keys, that shouldn't help.

Of course, the problem is that encoding schemes can rarely be kept secret. Everyone will get a copy of the algorithm. If there were no key, everyone who had a copy of the program would be able to decrypt anything encrypted with it. That wouldn't bode well for mass-market crypto products. A key enables the known good algorithms to be used in many places. So what do you do when you're faced with a product that says it uses Triple-DES encryption with no remembering of passwords required? Run away! DES and variants (like 3DES) depend on the secrecy of the key for their strength. If the key is known, the secrets can obviously be decrypted. Where is the product getting a key to work with if not from you? Off the hard drive, somewhere.

Is this better than if it just used a bad algorithm? This is probably slightly better if the files are to leave the machine, perhaps across a network. If they are intercepted there, they may still be safe. However, if the threat model includes people who have access to the machine itself it's pretty useless, since they can get the key as well. Cryptographers have become very good at determining what encoding scheme is being used and then decoding the messages. If you're talking about an encoding scheme that is embedded in some sort of mass-market product, forget the possibility of keeping it secret. Attackers will have all the opportunity they need to determine what the encoding scheme is.

If you run across a product that doesn't appear to require the exchange of keys of some sort and claims to have encrypted communications, think very hard about what you have. Ask the vendor a lot of questions of about exactly how it works. Think back to our earlier discussion about exchanging keys securely. If your vendor glosses over the key exchange portion of a product, and can't explain in painstaking detail how exactly the key exchange problem was solved, then you probably have an insecure product. In most cases, you should expect to have to program keys manually on the various communication endpoints.

Passwords Cannot Be Securely Stored on the Client Unless There Is Another Password to Protect Them

This statement about passwords specifically refers to programs that store some form of the password on the client machine in a client-server relationship. Remember that the client is always under the complete control of the person sitting in front of it. Therefore, there is generally no such thing as secure storage on client machines. What usually differentiates a server is that the user/attacker is forced to interact with it across a network, via what should be a limited interface. The one possible exception to all client storage being attackable is if encryption is used. This law is really a specific case of the previous one: "If a key isn't required, then you don't have encryption—you have encoding." Clearly, this applies to passwords just as it would to any other sort of information. It's mentioned as a separate case because passwords are often of particular interest in security applications. Every time an application asks you for a password, you should think to yourself, "How is it stored?" Some programs don't store the password after it's been used because they don't need it any longer—at least not until next time. For example, many Telnet and ftp clients don't remember passwords at all; they just pass them straight to the server. Other programs will offer to "remember" passwords for you. They may give you an icon to click on and not have to type the password.

How securely do these programs store your password? It turns out that in most cases, they can't store your password securely. As covered in the previous law, since they have no key to encrypt with, all they can do is encode. It may be a very complicated encoding, but it's encoding nonetheless, because the program has to be able to decode the password to use it. If the program can do it, so can someone else.

This one is also universal, though there can be apparent exceptions. For example, Windows will offer to save dial-up passwords. You click the icon and it logs into your ISP for you. Therefore, the password is encoded on the hard drive somewhere and it's fully decodable, right? Not necessarily. Microsoft has designed the storage of this password around the Windows

login. If you have such a saved password, try clicking **Cancel** instead of typing your login password the next time you boot Windows. You'll find that your saved dial-up password isn't available, because Windows uses the login password to unlock the dial-up password. All of this is stored in a .pwl file in your Windows directory.

Occasionally, for a variety of reasons, a software application will want to store some amount of information on a client machine. For Web browsers, this includes cookies and, sometimes, passwords. (The latest versions of Internet Explorer will offer to remember your names and passwords.). For programs intended to access servers with an authentication component, such as Telnet clients and mail readers, this is often a password. What's the purpose of storing your password? So that you don't have to type it every time.

Obviously, this feature isn't really a good idea. If you've got an icon on your machine that you can simply click to access a server, and it automatically supplies your username and password, then anyone who walks up can do the same. Can they do anything worse than this? As we'll see, the answer is yes.

Let's take the example of an e-mail client that is helpfully remembering your password for you. You make the mistake of leaving me alone in your office for a moment, with your computer. What can I do? Clearly, I can read your mail easily, but I'll want to arrange it so I can have permanent access to it, not just the one chance. Since most mail passwords pass in the clear (and let's assume that in this case that's true), if I had a packet capture program I could load onto your computer quickly, or if I had my laptop ready to go, I could grab your password off the wire. This is a bit more practical than the typical monitoring attack, since I now have a way to make your computer send your password at will.

However, I may not have time for such elaborate preparations. I may only have time to slip a diskette out of my shirt and copy a file. Perhaps I might send the file across your network link instead, if I'm confident I won't show up in a log somewhere and be noticed. Of course, I'd have to have an idea what file(s) I was after. This would require some preparation or research. I'd have to know what mail program you typically use. But if I'm in your office, chances are good that I would have had an opportunity

to exchange mail with you at some point, and every e-mail you send to me tells me in the message headers what e-mail program you use.

What's in this file I steal? Your stored password, of course. Some programs will simply store the password in the clear, enabling me to read it directly. That sounds bad, but as we'll see, programs that do that are simply being honest. In this instance, you should try to turn off any features that allow for local password storage if possible. Try to encourage vendors not to put in these sorts of "features."

Let's assume for a moment that's not the case. I look at the file and I don't see anything that looks like a password. What do I do? I get a copy of the same program, use your file, and click **Connect**. Bingo, I've got (your) mail. If I'm still curious, in addition to being able to get your mail I can now set up the packet capture and find your password at my leisure.

It gets worse yet. For expediency's sake, maybe there's a reason I don't want to (or can't) just hit **Connect** and watch the password fly by. Perhaps I can't reach your mail server at the moment, because it's on a private network. And perhaps you were using a protocol that doesn't send the password in the clear after all. Can I still do anything with your file I've stolen? Of course.

Consider this: without any assistance, your mail program knows how to decode the password and send it (or some form of it). How does it do that? Obviously it knows something you don't, at least not yet. It either knows the algorithm to reverse the encoding, which is the same for every copy of that program, or it knows the secret key to decrypt the password, which must be stored on your computer.

In either case, if I've been careful about stealing the right files, I've got what I need to figure out your password without ever trying to use it. If it's a simple decode, I can figure out the algorithm by doing some experimentation and trying to guess the algorithm, or I can disassemble the portion of the program that does that and figure it out that way. It may take some time, but if I'm persistent, I have everything I need to do so. Then I can share it with the world so everyone else can do it easily.

If the program uses real encryption, it's still not safe if I've stolen the right file(s). Somewhere that program must have also stored the decryption

key; if it didn't it couldn't decode your password, and clearly it can. I just have to make sure I steal the decryption key as well.

Couldn't the program require the legitimate user to remember the decryption key? Sure, but then why store the client password in the first place? The point was to keep the user from having to type in a password all the time.

Notes from the Underground…

Vigilance is Required Always!

Much discussion has been raised recently about the number of attacks that occur and the rapid deployment and proliferation of malicious codes and attacks. Fortunately, most of the attacks are developed to attack vulnerabilities in operating system and application code that have been known for some time. As we saw this year, many of the Code Red attacks and the variants that developed from them were attacking long-known vulnerabilities in the targeted products. The sad thing (and this should be embarrassing both professionally and personally) was the obvious number of network administrators and technicians who had failed to follow the availability of fixes for these systems and keep them patched and up-to-date. No amount of teaching, and no amount of technical reference materials can protect your systems if you don't stay vigilant and on top of the repairs and fixes that are available.

In Order for a System to Begin to Be Considered Secure, It Must Undergo an Independent Security Audit

Writers know that they can't proofread their own work. Programmers ought to know that they can't bug-test their own programs. Most software companies realize this, and they employ software testers. These software

testers look for bugs in the programs that keep them from performing their stated functions. This is called *functional testing*.

Functional testing is vastly different from security testing, although on the surface, they sound similar. They're both looking for bugs, right? Yes and no. Security testing (which ought to be a large superset of functionality testing) requires much more in-depth analysis of a program, usually including an examination of the source code. Functionality testing is done to ensure that a large percentage of the users will be able to use the product without complaining. Defending against the average user accidentally stumbling across a problem is much easier than trying to keep a knowledgeable hacker from breaking a program any way he can.

Even without fully discussing what a security audit is, it should be becoming obvious why it's needed. How many commercial products undergo a security review? Almost none. Usually the only ones that have even a cursory security review are security products. Even then, it often becomes apparent later on that they didn't get a proper review.

Notice that this law contains the word "begin." A security audit is only one step in the process of producing secure systems. You only have to read the archives of any vulnerability reporting list to realize that software packages are full of holes. Not only that, but we see the same mistakes made over and over again by various software vendors. Clearly, those represent a category in which not even the most minimal amount of auditing was done.

Probably one of the most interesting examples of how auditing has produced a more secure software package is OpenBSD. Originally a branch-off from the NetBSD project, OpenBSD decided to emphasize security as its focus. The OpenBSD team spent a couple of years auditing the source code for bugs and fixing them. They fixed any bugs they found, whether they appeared to be security related or not. When they found a common bug, they would go back and search all the source code to see whether that type of error had been made anywhere else.

The end result is that OpenBSD is widely considered one of the most secure operating systems there is. Frequently, when a new bug is found in NetBSD or FreeBSD (another BSD variant), OpenBSD is found to be not vulnerable. Sometimes the reason it's not vulnerable is that the problem

was fixed (by accident) during the normal process of killing all bugs. In other cases, it was recognized that there was a hole, and it was fixed. In those cases, NetBSD and FreeBSD (if they have the same piece of code) were vulnerable because someone didn't check the OpenBSD database for new fixes (all the OpenBSD fixes are made public).

Security through Obscurity Does Not Work

Basically, "security through obscurity" (known as STO) is the idea that something is secure simply because it isn't obvious, advertised, or interesting. A good example is a new Web server. Suppose you're in the process of making a new Web server available to the Internet. You may think that because you haven't registered a Domain Name System (DNS) name yet, and because no links exist to the Web server, you can put off securing the machine until you're ready to go live.

The problem is, port scans have become a permanent fixture on the Internet. Depending on your luck, it will probably be only a matter of days or even hours before your Web server is discovered. Why are these port scans permitted to occur? They aren't illegal in most places, and most ISPs won't do anything when you report that you're being portscanned.

What can happen if you get portscanned? The vast majority of systems and software packages are insecure out of the box. In other words, if you attach a system to the Internet, you can be broken into relatively easily unless you actively take steps to make it more secure. Most attackers who are port scanning are looking for particular vulnerabilities. If you happen to have the particular vulnerability they are looking for, they have an exploit program that will compromise your Web server in seconds. If you're lucky, you'll notice it. If not, you could continue to "secure" the host, only to find out later that the attacker left a backdoor that you couldn't block, because you'd already been compromised.

Worse still, in the last year a number of worms have become permanent fixtures on the Internet. These worms are constantly scanning for new victims, such as a fresh, unsecured Web server. Even when the worms are in their quietest period, any host on the Internet will get a couple of

probes per day. When the worms are busiest, every host on the Internet gets probes every few minutes, which is about how long an unpatched Web server has to live. Never assume it's safe to leave a hole or to get sloppy simply because you think no one will find it. The minute a new hole is discovered that reveals program code, for example, you're exposed. An attacker doesn't have to do a lot of research ahead of time and wait patiently. Often the holes in programs are publicized very quickly, and lead to the vulnerability being attacked on vulnerable systems.

Let me clarify a few points about STO: Keeping things obscure isn't necessarily bad. You don't want to give away any more information than you need to. You can take advantage of obscurity; just don't rely on it. Also, carefully consider whether you might have a better server in the long run by making source code available so that people can review it and make their own patches as needed. Be prepared, though, to have a round or two of holes before it becomes secure.

How obscure is obscure enough? One problem with the concept of STO is that there is no agreement about what constitutes obscurity and what can be treated like a bona fide secret. For example, whether your password is a secret or is simply "obscured" probably depends on how you handle it. If you've got it written down on a piece of paper under your keyboard and you're hoping no one will find it, I'd call that STO. (By the way, that's the first place I'd look. At one company where I worked, we used steel cables with padlocks to lock computers down to the desks. I'd often be called upon to move a computer, and the user would have neglected to provide the key as requested. I'd check for the key in this order: pencil holder, under the keyboard, top drawer. I had about a 50 percent success rate for finding the key.)

It comes down to a judgment call. My personal philosophy is that all security is STO. It doesn't matter whether you're talking about a house key under the mat or a 128-bit crypto key. The question is, does the attacker know what he needs, or can he discover it? Many systems and sites have long survived in obscurity, reinforcing their belief that there is no reason to target them. We'll have to see whether it's simply a matter of time before they are compromised.

Summary

In this Appendix, we have tried to provide you with an initial look at the basic laws of security that we work with on a regular basis. We've looked at a number of different topic areas to introduce our concepts and our list of the laws of security. These have included initial glances at some concepts that may be new to you, and that should inspire a fresh look at some of the areas of vulnerability as we begin to protect our networks. We've looked at physical control issues, encryption and the exchange of encryption keys. We've also begun to look at firewalls, virus detection programs, and intrusion detection systems (IDSs), as well as modification of code to bypass firewalls, viruses, and IDSs, cryptography, auditing, and security through obscurity. As you have seen, not all of the laws are absolutes, but rather an area of work that we use to try to define the needs for security, the vulnerabilities, and security problems that should be observed and repaired as we can. All of these areas are in need of constant evaluation and work as we continue to try to secure our systems against attack.

Solutions Fast Track

Knowing the Laws of Security

- ☑ Review the laws.
- ☑ Use the laws to make your system more secure.
- ☑ Remember that the laws change.

Client-Side Security Doesn't Work

- ☑ Client-side security is security enforced solely on the client.
- ☑ The user always has the opportunity to break the security, because he or she is in control of the machine.
- ☑ Client-side security will not provide security if time and resources are available to the attacker.

You Cannot Securely Exchange Encryption Keys

without a Shared Piece of Information

- ☑ Shared information is used to validate machines prior to session creation.
- ☑ You can exchange shared private keys or use Secure Sockets Layer (SSL) through your browser.
- ☑ Key exchanges are vulnerable to man-in-the-middle (MITM) attacks.

Malicious Code Cannot Be 100 Percent Protected against

- ☑ Software products are not perfect.
- ☑ Virus and Trojan detection software relies on signature files.
- ☑ Minor changes in the code signature can produce a non-detectable variation (until the next signature file is released).

Any Malicious Code Can Be Completely Morphed to Bypass Signature Detection

- ☑ Attackers can change the identity or signature of a file quickly.
- ☑ Attackers can use compression, encryption, and passwords to change the look of code.
- ☑ You can't protect against every possible modification.

Firewalls Cannot Protect You 100 Percent from Attack

- ☑ Firewalls can be software or hardware, or both.
- ☑ The primary function of a firewall is to filter incoming and outgoing packets.
- ☑ Successful attacks are possible as a result of improper rules, policies, and maintenance problems.

Any IDS Can Be Evaded

- ☑ Intrusion detection systems (IDSs) are often passive designs.
- ☑ It is difficult for an attacker to detect the presence of IDS systems when probing.
- ☑ An IDS is subject to improper configuration and lack of maintenance. These conditions may provide opportunity for attack.

Secret Cryptographic Algorithms Are Not Secure

- ☑ Crypto is hard.
- ☑ Most crypto doesn't get reviewed and tested enough prior to launch.
- ☑ Common algorithms are in use in multiple areas. They are difficult, but not impossible, to attack.

If a Key Is Not Required, You Do Not Have Encryption—You Have Encoding

- ☑ This law is universal; there are no exceptions.
- ☑ Encryption is used to protect the encoding. If no key is present, you can't encrypt.
- ☑ Keys must be kept secret, or no security is present.

Passwords Cannot Be Securely Stored on the Client Unless There Is Another Password to Protect Them

- ☑ It is easy to detect password information stored on client machines.
- ☑ If a password is unencrypted or unwrapped when it is stored, it is not secure.
- ☑ Password security on client machines requires a second mechanism to provide security.

In Order for a System to Begin to Be Considered Secure, It Must Undergo an Independent Security

Audit

- ☑ Auditing is the start of a good security systems analysis.
- ☑ Security systems are often not reviewed properly or completely, leading to holes.
- ☑ Outside checking is critical to defense; lack of it is an invitation to attack.

Security through Obscurity Does Not Work

- ☑ Hiding it doesn't secure it.
- ☑ Proactive protection is needed.
- ☑ The use of obscurity alone invites compromise.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form. You will also gain access to thousands of other FAQs at ITFAQnet.com.

Q: How much effort should I spend trying to apply these laws to a particular system that I'm interested in reviewing?

A: That depends on what your reason for review is. If you're doing so for purposes of determining how secure a system is so that you can feel comfortable using it yourself, then you need to weigh your time against your threat model. If you're expecting to use the package, it's directly reachable by the Internet at large, and it's widely available, you should probably spend a lot of time checking it. If it will be used in some sort of back-end system, if it's custom designed, or if the system it's on is protected in some other way, you may want to spend more time elsewhere.

Similarly, if you're performing some sort of penetration test, you will have to weigh your chances of success using one particular avenue of attack versus another. It may be appropriate to visit each system that you can attack in

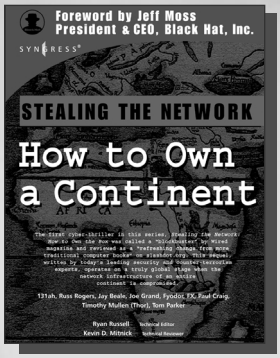
turn, and return to those that look more promising. Most attackers would favor a system they could replicate in their own lab, returning to the actual target later with a working exploit.

Q: How secure am I likely to be after reviewing a system myself?

A: This depends partially on how much effort you expend. In addition, you have to assume that you didn't find all the holes. However, if you spend a reasonable amount of time, you've probably spotted the low-hanging fruit—the easy holes. This puts you ahead of the game. The script kiddies will be looking for the easy holes. Even if you become the target of a talented attacker, the attacker may try the easy holes, so you should have some way of burglar-alarming them. Since you're likely to find something when you look, and you'll probably publish your findings, everyone will know about the holes. Keep in mind that you're protected against the ones you know about, but not against the ones you don't know about. One way to help

Syngress: *The Definition of a Serious Security Library*

Syn·gress (sin-gres): *noun, sing*. Freedom from risk or danger; safety. See *security*.



AVAILABLE NOW
order @
www.syngress.com

Stealing the Network: How to Own a Continent

131ah, Russ Rogers, Jay Beale, Joe Grand, Fyodor, FX, Paul Craig, Timothy Mullen (Thor), Tom Parker, Ryan Russell, Kevin D. Mitnick

The first book in the "*Stealing the Network*" series was called a "blockbuster" by Wired magazine, a "refreshing change from more traditional computer books" by Slashdot.org, and "an entertaining and informative look at the weapons and tactics employed by those who attack and defend digital systems" by Amazon.com. This follow-on book once again combines a set of fictional stories with real technology to show readers the danger that lurks in the shadows of the information security industry... Could hackers take over a continent?

ISBN: 1-931836-05-1

Price: $49.95 US $69.95 CAN

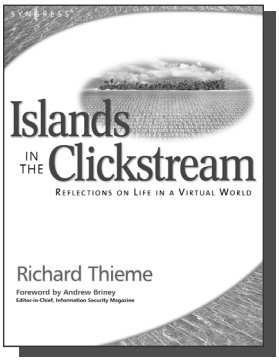
AVAILABLE NOW
order @
www.syngress.com

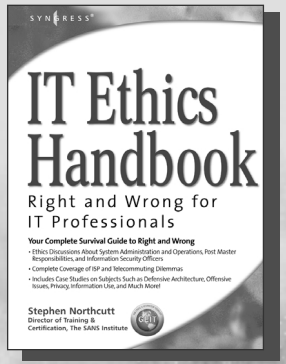
Richard Thieme's Islands in the Clickstream: Reflections on Life in a Virtual World

Richard Thieme is one of the most visible commentators on technology and society, appearing regularly on CNN radio, TechTV, and various other national media outlets. He is also in great demand as a public speaker, delivering his "Human Dimension of Technology" talk to over 50,000 live audience members each year. *Islands in the Clickstream* is a single volume "best of Richard Thieme."

ISBN: 1-931836-22-1

Price: $29.95 US $43.95 CAN





AVAILABLE NOW
order @
www.syngress.com

IT Ethics Handbook: Right and Wrong for IT Professionals

Stephen Northcutt

The final word on ethics and IT management from world-renowned security expert Stephen Northcutt, former Chief for Information Warfare at the Ballistic Missile Defense Organization and current Director of Training and Certification for the SANS Institute. This is not a textbook. Rather, it provides specific guidelines to system administrators, security consultants, and programmers on how to apply ethical standards to day-to-day operations.

ISBN: 1-931836-14-0

Price: $49.95 US $69.95 CAN

SYNGRESS®