;login:

THE MAGAZINE OF USENIX & SAGE
February 2001 • volume 26 • number 1

inside:

RIK FARROW:
MUSINGS

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

musings

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of *UNIX System Security* and *System Administrator's Guide to System V*.

<rik@spirit.com>

Thinking outside the box. That is how security exploits get created, and what software writers most often forget about. Several months ago, traceroute, a set-user-id root program, was exploited by calling the source route option flag twice. Who would have thought that anyone would use the same command line option twice? Certainly not the author of traceroute, who was really concerned about creating a tool that could show the route to a destination, or where the route prematurely ended.

Thinking outside the box doesn't have to be particularly deep thinking. Web sites that include the purchase price of items in the URL make it easy to change the price, simply by editing the URL (almost point-and-click). Also exploitable are firewalls that include a backdoor for vendor support and use sniffable passwords for root access. No one was supposed to know about port 3000, and besides, popular password sniffers only listened to low-numbered ports. This particular hole was fixed many years ago – it just still amazes me that a well-known (at the time) firewall vendor would do such a thing.

Another way of thinking outside the box is through "misuses" of networking protocols. Now, really, there is no such thing as a misuse of a protocol. Protocols are conventions that permit communication, usually between consenting clients and servers. For example, when you use a Web browser, it obeys the conventions found in either RFC 1945 or RFC 2068 to communicate with the Web server (HTTP versions 1 and 1.1). Essentially, the Web browser sends a request that includes a simple header to the server, and the server sends back a simple header that includes as its first item a result code, and possibly the requested item.

You can do more than request Web pages. You can execute code on the Web server through CGI, ASP, server-side includes, and other mechanisms, like Java servlets. But let's think outside the box for just a minute. Most organizations that have firewalls permit their users to roam the Web. In some instances, certain sites are blocked based on their names (and because these sites contain information not pertinent to work or in compliance with accepted morals at the organization). We can assume that with the exception of these cases, you can use a Web browser to connect to remote Web servers.

Well, then you can also use HTTP to tunnel through your firewall. For example, suppose your firewall does not permit you to telnet to your home computer. You could then download the HTTPTunnel (<*http://www.nocrew.org/software/httptunnel.html*>), forward the remote end to port 23, where your telnetd is listening, and use the client side of this tunnel (htc) to forward a local telnet connection through the tunnel. The data will be formatted as valid HTTP PUT requests and responses, and your firewall will happily let you use telnet – as long as it is embedded within the HTTPTunnel.

Before Checkpoint changed their defaults (with version 4.1), a fun thing to do was to set up a server listening at port 53, such as netcat that executed a shell, and connect to it through the firewall. Although port 53/TCP is supposed to be used for DNS, most firewalls do nothing to enforce the actual use of DNS, so you can connect to a shell, enter commands, and have the results sent back. This is almost too trivial. Only application gateways (sometimes called proxy servers, or, in the case of Checkpoint, security servers) actually check to see if the appropriate protocol is being used on a particular port. The HTTPTunnel will pass most application gateways, as it conforms to protocol specs in the RFCs for HTTP headers.

There is an even cuter trick someone wrote. You can use DNS requests to tunnel commands to a remote server. A posting on Slashdot describes a "new protocol" NSTX (Nameserver Transport) that permits you to use a special client to send compliant nameserver requests to a special server that can then execute commands and send back the results as if they were actual DNS replies (<*http://slashdot.org/articles/00/09/10/2230242.shtml*> and <*http://nstx.dereference.de/*> for the code). Unlike HTTPTunnel, NSTX is no longer under development.

You can use DNS requests to tunnel commands to a remote server.

Of course, the Web trick that has a lot of people upset these days comes from Microsoft. I think what caused the uproar was a paper on an MS Web server that described SOAP (Simple Object Access Protocol) as "a way to slip through firewalls." Unlike HTTPTunnel, SOAP is less a way to slip through firewalls, and more a new protocol for supporting remote procedure calls. SOAP requires a new header in the HTTP request line, SOAPMethodName, that includes as its argument a URN (Universal Resource Name). The name found here must also match the first sub-item found in the XML (Extensible Markup Language) sent as the body of the request. In SOAP, XML is used for data representation.

And, if anything has annoyed me more lately, it is XML. Such a squirrelly language, it can morph into anything the designer wants, while appearing to be harmless. Someday we will begin seeing XML exploits, but not yet. The day is closer, however, as Microsoft and VeriSign have announced PKI extensions for XML, XKMS.

If you really want to understand more about SOAP, you can read "A Young Person's Guide to XML" at <*http://msdn.microsoft.com/msdnmag/issues/0300/soap/soap.asp*>.

And if you ever have occasion to read Bugtraq these days (<*http://www.securityfocus.com*>, Forums), you will have heard of Ofir Arkin. Ofir has been studying the small differences in the ways that vendors implement ICMP, and has written a paper describing his researches (<*http://www.sys-security.com*>). Ofir has been digging at this for over a year now, and has forced the security community to pay more attention to ICMP. His recommendation is to block all ICMP packets at your firewall (presuming you have one), something that members of the IETF might shudder thinking about. Read his paper and you will begin to understand why.

Not just because of the ping of death, either, or the floods generated by smurf DoS attacks. Ofir has discovered ways of fingerprinting hosts even if you block ICMP packets going to those hosts (you must permit other IP packets to the target host). ICMP Time Exceeded errors can be used to identify certain operating systems by sending only one part of a fragmented packet.

But there is another reason why you might want to block ICMP – if you are paranoid enough. After all, people can use HTTP to tunnel out through your firewall, so why worry about ICMP? Because ICMP has also been used to tunnel information through firewalls.

I first encountered an ICMP tunnel through the gift of a friend who works at a university. The tools were called pinsh and ponsh, one a client, the other a server, and both communicated using ICMP ECHO_REPLIES (what you would normally receive in response to an ECHO_REQUEST, as generated by ping). ICMP ECHO packets may include an optional payload, and pinsh/ponsh used this payload to carry commands and the output of the commands back and forth.

If you permit any communications at all between your network and other networks, your network can leak data like a sieve.

This idea was not new, as daemon9 had already written about it in Phrack issue 49. Loki v2 (<*http://www.2600.com/phrack/p51-06.html*>) adds encryption and digital signatures to further disguise the tunneled data and to authenticate the requests (wouldn't want the wrong people using our tunnel). The code found here only works with Linux systems, and has not been maintained. But you get the idea.

That is, if you permit any communications at all between your network and other networks, your network can leak data like a sieve. Note that this usually requires the active participation of some internal user – unless you are using Windows. In that case, you might fall victim to a virus-installed Trojan, such as Subseven, that provides a simple way to remote control your Windows NT box. As most firewalls block all incoming connections, some Windows Trojans make an outgoing connection to an IRC server, join a particular channel, and wait for commands. This technique has also appeared in agents for DDoS attacks (Trinity, which you can read about in Sven Dietrich's paper in the LISA 2000 proceedings). At the very least, block outgoing connections to port 139 (used by SMB and Samba), because Microsoft OSes consider shared files as part of the trusted security context. In other words, an attacker can provide code that your system will run on a remote file share, and it will be trusted. You can prevent this by updating IE (often), and by blocking port 139/TCP outgoing.

Holy networks! Is there no end to this? Actually, I'd like to mention a very old hack by Marcus Ranum, performed to illustrate the leakiness of networks in general. Marcus tunneled NFS over UUCP (using email) just to make a point about leakiness of networks. If you really want to prevent data leaking from networks, you can neither connect them to any other network nor permit anyone to use modems. Oh, and you might also want to include a degaussing magnet, à la Neal Stephenson's Cryptonomicon, although you'll want to warn people using pacemakers, and wipe clean magnetic media as it leaves your site. Anyone have a writable CD handy?