RIK FARROW

musings

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

A CURIOUS CONFLUENCE OF EVENTS has prompted me to renew my acquaintance with NFS security. For some reason, I had allowed NFS to slip from my awareness some time ago. Perhaps it was because the problems facing those who wanted to use NFS securely seemed overwhelming. Just as likely, other new things that glittered and shimmered with expectation grabbed my attention, leaving the venerable NFS to sit ignored in a corner.

But only by me. Many organizations rely on NFS, including my friends at San Diego Supercomputing Center. Elsewhere in this issue, you should find a tale told by Abe Singer about the experience he and his fellow workers had with an intrusion that began in Spring of 2004. The intruder continued to abuse SDSC systems for weeks and, at some point, took advantage of old, known weaknesses in NFS to do so.

While in Atlanta teaching my class for LISA, I asked how many people were using NFS in their organizations and was surprised by the result. I guess I had been asking the same question in the wrong venues, because lots of people indicated that they were using NFS. That, coupled with listening to Brian Pawlowski's (Netapp.com) Invited Talk about the future of NFS, really got me interested.

The Past

NFS started out at UC Berkeley and then was adopted by Sun Microsystems. The concept of a network file system didn't start with NFS but had (at least) one earlier, significant implementation. Apollo Domain had a network file system, along with single sign-on, unified user accounts, and home directories supported via the network file systems. You could log on to any Apollo workstation and get your home environment. While this may sound familiar, the underlying architecture paid serious attention to security. Apollo also had signed patches that could be centrally installed by the system administrator. But I digress.

NFS security, ever since its beginning, had barely existed. I won't say it didn't exist; that would be unfair. And today, most organizations continue to use the earliest form of NFS security, not because nothing else exists, but because it is the lowest common denominator and works on anything that supports NFS.

In the past, NFS security took two forms: user identification, and IP address–based access control. Let's talk about user identification first.

The user identification is called AUTH_UNIX, a somewhat misleading name, as AUTH hints at authentication, when there really is none at all. In AUTH_UNIX, each Remote Procedure Call (RPC) request sent to an NFS server includes the requesting user's ID number (UID). The UID gets assigned to your account via your /etc/passwd entry associated with your login shell and is used to signify ownership of files by being included in file attributes (within inodes). You can see the UID and GIDs (group IDs) associated with files when you use the command ls -ln (the -n option suppresses the conversion of UIDs to names).

NFS includes this very same UID, along with up to 16 GIDs, in each RPC request. The UID and GIDs will then be used to determine access to files and directories on the NFS server. Perhaps at this point you have discerned why I don't call AUTH_UNIX authentication. While a UID may be used to identify a user, it is not in itself a form of authentication. Anyone who can manipulate the RPC request can insert any UID he or she wants. No authentication is required to do this. None at all.

A Curious Tool

Back in 1991, Leendert van Doorn wrote a tool he called the nfsshell. The nfsshell generates legal NFS client requests, with the optional feature that the user of nfsshell can insert the desired UID into any request. No special privilege is required to use nfsshell—that is, you can run nfsshell as an ordinary user and can easily masquerade as other users when accessing an NFS server.

Van Doorn, who now works for IBM (he is shown acting as an antenna in his signature page, http://www.research.ibm.com/people/l/leendert/), created nfsshell to demonstrate the insecurity of AUTH_UNIX. You can find a copy of nfsshell on Van Doorn's Web page, although you will need to find patches, and perhaps an include file, if you want to get this working on Linux.

I configured an old Solaris system (2.7) to act as an NFS server, then tested it using a Linux system. Then I fired up nfsshell and started experimenting. Nfsshell worked as advertised in that I could create a file on the NFS server that could only be read by some other user, change the UID with nfsshell to be the same as that user, and read that file. The interface to nfsshell is similar to FTP or smbclient in general outline, and not at all hard to use.

One saving grace to NFS that I should point out is that by default UID zero, or root, gets mapped to -2 (username "nobody," by tradition). So changing the UID to 0 in nfsshell usually works worse than picking on some other UID. Note that you can disable this safety feature on your NFS server by exporting or sharing directories and using the option anon=0. You *really* don't want to do this. You can use the root option with a list of hostnames if you really want to permit root access to specific systems. This feature was designed for diskless workstations, pretty rare beasts these days.

At this point, NFS users should be either experiencing shock or muttering to themselves, "I knew it was so." On a more practical note, you should be wondering what you can do to add real security to NFS. After all, this is a disaster, isn't it?

NFS also started out with host-based access control. That is, the person who configured the NFS server could limit which clients could mount an exported ("shared" in Solaris) file system. There have been loads of problems with this scheme, some involving bugs, others involving configuration errors, and many meaning that file systems were exported to the world. Dan Farmer and Wietse Venema's SATAN scanner (1995) alarmed the world when released, as well as alarming NFS administrators when the tool would announce file systems exported to the world.

Another problem with earlier versions of NFS is that they relied on UDP for transport. UDP was chosen because it is stateless and was quite a bit faster than TCP back in the early eighties. UDP is also trivial to spoof, making it easy to get around the host-based access control, which relies on the IP address of the client. NFS versions 3 and 4 support TCP instead of UDP as the transport mechanism, and you can include "tcp" in the options when you export or share file systems, forcing the use of TCP.

Host-based access control does not solve the problem of authentication. For that, real authentication, based on cryptography, is required.

Authentic

Even before Van Doorn's nfsshell appeared, people were concerned about the lack of authentication in NFS. Sun Microsystems developed AUTH_DH, where DH stands for Diffie-Hellman key exchange. (AUTH_DH used to be called AUTH_DES, but I am using its more modern label.) Instead of simply relying on a UID, AUTH_DH uses Diffie-Hellman to exchange session keys for each user. The session key is used to encrypt a pair of timestamps that are included in the RCP header. If the decryption fails, or the timestamp falls outside a five-minute window, then the request is considered unauthenticated.

AUTH_DH relies either on NIS to distribute public and private keys or on the manual distribution of the

/etc/publickey file to all NFS servers and clients. Every time a user changes her password, the public key file must be updated, so practical use of AUTH_DH seems to imply use of NIS as well. You can read more about using AUTH_DH in O'Reilly's *Managing NFS and NIS*.

Generic Security Services for RPC, or RPCSEC_GSS, represents the emerging alternative to AUTH_DH. GSS can support not only authentication but also integrity and privacy, and support for all three have been added to NFSv4 and in some cases are included in patches for older versions of NFS. GSS can use multiple security providers, with Kerberos v5 being the most common.

Setting up a Kerberos Domain is in many ways akin to setting up NIS servers, in that you need secure systems to run the Kerberos Distribution Center (KDC), as well as slave servers for backups. I had hoped to torture myself by setting up a KDC for this column but, fortunately, ran out of time. If you have a Kerberos infrastructure or have been moved by a real desire for NFS security to set one up, by adding krb5 to the export or share options, you can inform your NFS servers to permit access only to NFS clients that can handle Kerberos v5 authentication.

NFSv4 is not limited to Kerberos support for authentication; it is also beginning to include support for SSL-style authentication using LIPKEY/SPKM plugged into the lower layers of GSS instead of Kerberos. The advantage to SPKM is that NFS RPC requests can be authenticated by installing public key certificates for NFS servers or the CAs for those servers on client systems, similar to the way in which Web browsers include certificates for signing authorities. Using SSL, a session key gets generated and is used to sign RPC headers. Once again, you can add real authentication, but this time without setting up either NIS or Kerberos.

The downside to RPCSEC_GSS is that support for it is still very thin. Sun Microsystems has both clients and servers, and Kerberos support is included in newer versions of HP/UX and with AIX 5.2 (with a requested addition). But support in the Linux and BSD world is not quite there yet. You can visit http://www.linux-nfs.org and volunteer your efforts (or other forms of assistance) or check out CITI for FreeBSD support (http://www.citi.umich.edu/u/rees/). You can also read more about the state of NFS security in Michael Eisler's presentation about it: http://www.nfsconf.com/pres03/eisler.pdf.

There are other things you should do when using NFS. When exporting directories, it is wise to disable the use of set-user-ID and device files on the exporting server. The options nosuid and nodev in the export or share statement handle this and prevent a miscreant who has gotten root on a server from creating an SUID shell that will work on all systems that have mounted the file system. Note that clients can use the same flags when mounting remote file systems, disabling the use of SUID and device files from a possibly compromised remote system.

You might think that perhaps CIFS, Microsoft's Common Internet File System, would be a more secure choice. While it is true that CIFS (supported as Samba on UNIX systems) does include real authentication, it is not the same as NFS. For one thing, CIFS is oriented toward added file shares for one user at a time. That is, instead of mounting a file system for use by all users, you instead mount a file share using a username and password for one user. CIFS does not send plaintext passwords across the network (unless misconfigured), but the challenge-response pairs it does use can be cracked and passwords guessed.

Microsoft responded to this threat by beginning to support (you guessed it) Kerberos v5 with Windows 2000. Note that MS Kerberos v5 has some proprietary extensions added to each ticket that are only of use (or interest to) Microsoft systems. While MS Kerberos could be used to support UNIX systems (as the additional cruft will be ignored), the reverse is not true. Do you really expect some self-respecting UNIX system to understand the wacky, variable-length data structures that Microsoft uses as Security Identifiers (SIDs)? Perhaps some day.

So, sure, you can use CIFS instead of NFS. But really, NFS with Kerberos support is just as secure, and more, uh, UNIXy as well.

I wish I could say more, especially after having managed to disturb at least some of my readers, but that is the state of the art today.