;login:
THE MAGAZINE OF USENIX & SAGE
November 2001 • Volume 26 • Number 7

Special Focus Issue: Security
Guest Editor: Rik Farrow

inside:

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

no plaintext passwords

by Abe Singer

Abe Singer is a computer security manager at the San Diego Supercomputer Center, and occasional consultant and expert witness. His current work is in security measurement and security "for the life of the Republic."

abe@SDSC.EDU

Introduction

Compromise of a user password is one of the most difficult intrusions to detect. Historically it has been difficult or impossible to avoid transmission of passwords in the clear. But the technology now exists to make this possible, albeit not trivially. The San Diego Supercomputer Center (SDSC) has managed to eliminate plaintext password transmission, while continuing to deliver services to a widely distributed user base. While it took some technical effort, overcoming the human hurdles proved to be more challenging. This article discusses what solutions we provided and how we managed to do it without annoying too many people. We have actually added value to the environment, instead of reducing it.

At SDSC, we have to deal with some interesting issues of scale. We have thousands of users and very few support staff. We have a wide variety of operating systems, high-speed networks and high-performance storage systems. Our users expect to be able to move large amounts of data (terabytes) around, in a reasonable amount of time. They want to do streaming applications, grid computing, and stuff that has not yet been invented. In addition to providing computing resources to researchers, we have people doing research in high-performance computing, networking, and storage. Unlike many places, most of our users do not work inside networks that we control. They are spread all over the planet and work for different institutions. This means that our infrastructure must scale outside of our "trusted" networks.

Because of the nature of our users, and the work done within and outside the Center, we cannot (and do not want to) mandate homogeneity such as "everyone must use Outlook for email." Instead, we focus on supporting protocols, and let the users pick their clients. We attempt to provide reasonable support for the applications that our users are already using, instead of requiring them to use the one(s) that we've decided are easy to support. Oh, and by the way, we have not had a root-level compromise (that we are aware of) on our managed systems in over two years.

How do we do it? Mostly through the following:

* Strong configuration management
* Patch early, patch often
* Strong authentication, and no plaintext passwords, anywhere
* Simple, but strong, access control between "trusted" and "untrusted" networks

We have managed to turn off plaintext passwords and continue to provide support for almost all of the applications our users have. Additionally, we will provide services for applications that we don't support. For instance, we provide IMAP over SSL service, and support Netscape Mail and Outlook clients. However, if a user has another client that speaks IMAPS, they are welcome to use it. We just won't help them with problems with their client.

The result is that we have an environment where users can get their work done from anywhere in the world. They can use the software that they need and read their email with the clients that they like, and we have improved the security of our systems at the same time.

. . . most switches behave like hubs when their MAC tables are overloaded

Background

Most of the commonly used TCP protocols use plaintext passwords: telnet, the r-commands, ftp, pop, imap, and HTTP basic authentication. Other protocols can use either plaintext or encrypted passwords but may use plaintext passwords by default.

Effective access control requires strong authentication. This means using authentication mechanisms which cannot be easily bypassed or subverted through eavesdropping, cryptanalysis, or brute-force attack.[1]

An authentication scheme that is highly resistant to brute-force attack or cryptanalysis can be fundamentally useless if the password can be intercepted. Protection of passwords on hosts is reasonably well implemented: both UNIX and Windows provide encrypted storage of passwords and prevent exposure of the passwords to the users. However, many network protocols transmit these same passwords in plaintext.

Why is this a problem? Primarily because plaintext passwords can be easily intercepted via a sniffer. There are dozens of sniffer programs available.[2] Some sniffers are smart enough to filter out just the usernames and passwords, and produce username, remote host, and password in an easy-to-read format.

As mentioned above, we have not had a root-level compromise on our managed systems in over two years. But we do have some networks with systems managed by users or other groups. We have had compromises on those systems, and occasionally we help investigate intrusions on other systems. Most of the intrusions we've seen include the use of a sniffer.

An intruder may have any number of motives for breaking into a system: running an IRB "bot," setting up a "warez" site, or using the system as a cutout to attack other systems, for example. The intruder typically installs a rootkit, and the rootkit almost always includes a sniffer. Even when sniffing is not an intruder's primary motive, the sniffer is an opportunistic attempt to compromise user accounts on other systems. Since users often use the same password on multiple systems, an intruder will try the username and password on various machines, even at different sites, to see what they can log into.

In one case, a user burned passwords to three different sites, including ours. The user had set up their own system (on the "user-managed" network) because they supposedly needed to run their own FTP server. They would routinely telnet into the system, and from there ssh into our site and the two others. Eventually their system was compromised (due to an unpatched vulnerability). The intruder used the system to run a bot but also installed a sniffer. When we found the sniffer log, we saw several usernames and passwords into multiple sites. We notified the other sites involved and investigated our managed machines to determine whether or not the intruder had actually used the passwords (apparently not).

"Switched" networks are not immune to sniffing. Switches sometimes leak information. Some switches are not fully switched but are really "switching hubs," where groups of ports share data exactly like a hub. Most importantly, most switches behave like hubs when their MAC tables are overloaded.[3]

One of the big problems with password compromises is that they are difficult to detect. Since the intruder logs in with a legitimate username and password, they are successfully authenticated and look like a legitimate user to the system. A user account compromise can go undetected for months – in one case we know of, a compromise

went undetected for two years! Some detection is possible using user profiling, but this is cumbersome and inaccurate. We believe that efforts are better spent at eliminating the opportunity for interception in the first place.

The more effective solution is to either encrypt the password in transmission or authenticate without password transmission.

The Rollout

About three years ago, SDSC began turning off most plaintext password services. A year ago we turned off the last, with the exception of a few older systems that we haven't yet updated. (The long time frame was partly due to a lack of technology and partly due to the need to make sure that users were able to make the transition.) These systems allow plaintext within our trusted networks, but not outside.

We began by enabling SSH and Kerberos services. Users had the option of using Kerberized clients or SSH. FTP was enabled via tunneled SSH sessions. We bought 5,000 copies of SecureCRT and several hundred copies of F-Secure SSH for Macintosh to distribute to users who needed it.

We then announced that plaintext access would be cut off in nine months. We notified all user via email and mentioned the change in the message-of-the-day and in banners. All the messages included links to Web pages for information on how to get software and how to use it. We also periodically sent out reminders.

On the scheduled date, we turned off access to Telnet, rlogin, and FTP. We did this by changing TCP-wrappers to deny access and display a banner with an explanation and a URL for more information. Most of our users had already switched. Some of them had not, but as soon as they tried to log in and saw the rejection message, they had little choice but to make the appropriate transition. Very few called our help desk, as they sheepishly realized that we had given them plenty of notice. A few (20 or so) did call. Most of them had some problems understanding. One user, when asked if he had read seen our notices, responded, "I never read those things. They never say anything useful."

One user, when asked if he had read seen our notices, responded, "I never read those things. They never say anything useful"

We implemented email solutions as they became available. About a year and a half ago, we drew a matrix of all the email clients we had to support, the non-plaintext authentication mechanisms they supported, and servers implementing the same. We found that we could turn off plaintext access to email using a combination of IMAPS, POPS, APOP, KPOP, and NFS access for mail-reading from managed UNIX systems. We also found a Web-mail solution (IMHO Webmail)[4] to provide users with access to their email from any SSL-capable Web browser.

Six months later, after appropriate announcements and lead time, we turned off plaintext email services.

We have also rolled out SecureFTP,[5] sftp, and are evaluating Web-based access to user directories.[6]

The Technology

We describe here the various solutions we have implemented, with some tips based on our experience. We are not providing a tutorial on how to implement POP or IMAP servers. Rather, we discuss what we use to encrypt passwords on these services. References are provided for details of implementation.

Kerberos is a mature, well-reviewed, open protocol, having been around about 15 years.

Interactive Access

For interactive access, we support Kerberos[7] and SSH.[8]

Kerberos works very well. Installing and configuring Kerberos is not trivial, but it is very easy to use. Kerberos provides strong authentication (both user and host) without transmitting passwords, and can provide encrypted data transmission. It scales very well (our KDCs are Sparc5s), has low overhead, and provides redundancy for failover and remote administration tools. Kerberos is a mature, well-reviewed, open protocol, having been around about 15 years. See note 7 for a detailed source of information on running and installing Kerberos.

SSH is a replacement for Telnet which provides interactive shell access, rcp-like file copying, and the ability to tunnel other protocols across encrypted streams. All data streams in SSH are encrypted.

We are currently supporting version 1 SSH because, until recently, there were not version 2 clients for all of the platforms we have to support (Windows, UNIX/Linux, Macintosh). We support Kerberos authentication for SSH sessions.

We also support RSA public keys for authentication via SSH. This requires a user to generate a public-key pair, store the private key on their client machine(s), and install the public key on the server(s). We have mixed feelings about this option, as it requires users to keep their private key secure, and users are not known for being good at keeping data secure.

A source for information on various SSH clients can be found in reference 8.

Email

We support a variety of services for email. Our users have Eudora, Netscape Mail, Outlook, and others.

Our users have to be able to read and send mail from any location. However, we do not want to be an open relay for the entire world. Our solution involves authenticated SMTP over SSL, IMAP over SSL (IMAPS), APOP, POP XTND XMIT, HTTPS, POP over SSL, and KPOP.[9]

Here is what we support:

	Reading Mail		*Sending Mail*	
Client	Protocol	*Daemon*	*Protocol*	*Daemon*
Eudora	APOP	qpopper	POP XTND XMIT	Sendmail, qpopper
Outlook	IMAP/SSL	UW imapd, sslwrap	AUTH SMTP/SSL	Sendmail
Netscape	IMAP/SSL	UW imapd, sslwrap	AUTH SMTP/SSL	Sendmail
Webmail	HTTPS, IMAP	Roxen,IMHO, imapd	HTTPS	Sendmail, Roxen, IMHO
Mutt, Elm, Pine	NFS		SMTP	
Eudora	KPOP	qpopper		
Outlook	POP/SSL	qpopper, sslwrap		

IMAPS and POPS are implemented using sslwrap.[10] This is almost trivial to do.

We have a centralized mail hub running Sendmail. We will not relay mail for machines outside of our network, without authentication. However, users outside our network who need to send mail have the option of using either authenticated SMTP over SSL, or the XTND XMIT option of POP. Netscape Mail and Outlook support the former, Eudora supports the latter.

For UNIX mail clients (pine, elm, mutt) on internal hosts, we provide NFS access to incoming mail folders. We only allow NFS on "trusted" networks, which are the networks which only have hosts that we manage. Since users have to use Kerberos or SSH to access these hosts, they have already used a strong authentication method to access the system.

sslwrap is a relatively simple way to encrypt a TCP-based service. To sslwrap a service, first configure the service to accept connections only on the loopback interface (127.0.0.1). This can be done easily with TCP-wrappers (and you should be TCP-wrapping your services anyway). Next, place an entry in inetd for the secure version of that service (e.g., IMAP uses port 143, IMAPS uses port 993).[11]

The inetd.conf entry for imap and imaps looks like this:

```
imap    stream  tcp nowait  root    /usr/local/etc/tcpd  /usr/local/etc/imapd
imaps   stream  tcp nowait  nobody  /usr/local/etc/tcpd  /usr/local/etc/imapsd
```

imapsd is a simple shell script that looks like this:

```
/usr/local/etc/sslwrap -cert /usr/local/certs/ssl-imap.pem \
  -CAfile /usr/local/certs/ca-cert.pem -port 143
```

The TCP-wrapper configuration for these services looks like this:

```
imapd: 127.0.0.1: allow
imapd: ALL: rfc931: DENY
imapsd : ALL : rfc931 : ALLOW
```

The ssl-wrapper negotiates an encrypted session on the "secure" port, then connects through the loopback device to the original unencrypted service. The remote client gets an encrypted session, and the local client does not require any modification. The same can be done to implement POPS with any POP server.

KPOP is not trivial to configure, and Eudora only supports version 4 of Kerberos. The server must be compiled with Kerberos libraries, and is run on an alternate port with a command line option to enable Kerberos authentication. We have not had many users making use of KPOP. Refer to the qpopper documentation for instructions on implementation.[12]

APOP uses a challenge-response password hashing mechanism to avoid transmitting passwords in the clear. When a client connects to the server, the server presents a challenge string. The client hashes the user's password with that challenge, and returns the hash. The server authenticates the client by comparing that hash with its own hash of the password. In order to implement APOP, qpopper has to keep clear-text copies of user passwords in its own database, by default /etc/pop.auth. Tools are provided for managing this password database.

The XTND XMIT feature of qpopper allows mail delivery through the pop server. The pop server in turn delivers the mail by calling Sendmail locally. This allows us to pre-

sslwrap is a relatively simple way to encrypt a TCP-based service.

. . . the scp protocol has a two-gigabyte file limit, which is problematic for some of our users

vent open mail relaying on our mail server and still enable users to send mail through their POP clients.

XTMD XMIT also has to be configured on the client. For Windows Eudora clients, this requires editing a .ini file. For Macintosh clients, it requires that the Esoteric Settings plug-in be installed.[13]

Authenticated SMTP over SSL implements the SMTP AUTH feature,[14] using SSL to encrypt the data stream. We implement this feature using Open Sendmail,[15] with SASL[16] and the entropy-gathering daemon.[17] With Sendmail, the server must be dedicated to authenticated SMTP, so we run a separate mail host just for authenticated relaying.[18] A source for detailed implementation instructions is in this reference.

For users who insist on using other mail clients which do not support the above protocols, we also support SSH-port forwarding.[19] Our POP and IMAP servers allow plaintext authentication via the loopback device. A user can establish an SSH connection to our mail server and then tunnel any mail client they want. SSH tunneling can be tricky to do, but it does work.

Finally, we have some users who may not have access to a mail client. They may be using another person's machine, in a terminal room in a conference, or at an "Internet cafe," for example. In addition to the various client support above, we provide a Web-based mail client called IMHO (see note 4). The client runs under the Roxen[20] Web server, and talks IMAP to the mail server via localhost.

File Transfer

For file transfers, we support scp through SSH (version 1), sftp (through SSH version 2), KFTP, and SecureFTP (see note 5).

scp is supported through the SSHD server. However, the scp protocol has a two-gigabyte file limit, which is problematic for some of our users.

sftp uses a separate binary which is invoked by the SSHD server. An entry in the SSHD configuration file points to the sftp binary.

KFTP is a Kerberized version of FTP. It uses Kerberos authentication on the command channel, but data transfers remain plaintext. This is considered a feature by some of our users.

Many of our users are not concerned with data confidentiality, only data integrity. Most are researchers using open or published data. Many of them transfer large amounts of data – sometimes terabytes. In these cases, the overhead of encryption creates too large a performance problem. At one point in time, our users found that encryption increased transmission time by a factor of four.

SecureFTP is an ssl-wrapped FTP server, with command line clients and a Java-based client that can be run from a Web browser. The command channel is encrypted, which protects passwords during transmission. Like KFTP, the data channel is left unencrypted. Any FTP server can be ssl-wrapped, and the Java client can be run from any operating system (which supports Java). See reference 5 for where to find details.

File Sharing

We currently do not allow file sharing outside of our trusted networks. Within our networks, we provide NFS for UNIX systems, Netbios/SMB for Windows systems, and

AppleTalk for Macintosh. We have a handful of centralized file servers where all user data lives – home directories, project areas, etc.

NFS does not provide user authentication. We only export to trusted hosts, and the NFS server will only talk to trusted networks.

We use Samba to provide file sharing for the Windows systems. We do this so that we can export the same data to the Windows systems as we do the UNIX systems. Samba can authenticate using UNIX passwords, its own password file, or through a Windows PDC, but password encryption is only available for the latter two methods. We authenticate Samba users against a PDC, which is also used to authenticate Windows logins. In order to match file ownership properly against a UNIX system, Samba requires that the UNIX usernames match the Windows usernames, or you must manually maintain an equivalence list.

We use Netatalk[21] to provide AppleTalk file shares similar to how we provide Windows file shares. Currently this uses plaintext passwords, and is restricted to our Macintosh networks.

Other

We also maintain an anonymous-only FTP site. It is configured anonymous-only so that users aren't tempted to use the server for file transfers. All users have an incoming and outgoing folder that they can use anonymously. This provides a fallback method of transferring files when authenticated access is unavailable.

PGP software is available for those who wish to use it. We provide version 2 and version 5+ software.[22]

Weaknesses

Our system is not (yet :-]) perfect. There are some known weaknesses, most of which will be addressed over time.

Our biggest problem is what we call "two-hopping." A user at a remote site, who does not have an SSH client on their desk (machine A), will telnet to another system which does have SSH (machine B), and then SSH into our site. The password into our site is intercepted by a sniffer between A and B. The intruder then uses the password to SSH into our site. Sometimes the less-than-clueful user telnets through several machines before ssh-ing into ours. In some cases, we have alerted the user and site administration, and changed their passwords, only to have it happen again a few days later. When asked to install SSH, the user complains that it is too hard, or they don't have the fifteen dollars for a site-licensed copy! (and now there are free versions of SSH available)

Our biggest problem is what we call "two-hopping."

Macintosh file sharing via AppleTalk currently sends passwords in plaintext. As mentioned above, we expect to move to DoubleTalk and Samba.

Several of the services we implement (e.g., APOP and Kerberos) require access to stored plaintext passwords (encrypted on disk with a shared key). While this is less than desirable, the hosts on which these passwords are stored are within our control and are kept relatively secure. The risk to us is much less than a user storing a plaintext password on their home computer.

The anonymous-only feature of our FTP server (wu-ftpd) does not reject the login until after the password is provided. The result is that users who don't know the server is anonymous-only will "burn" their password the first time they try to use the service.

REFERENCES

1. B. Schneier, *Applied Cryptography*, 2nd ed. (John Wiley and Sons, Inc., 1996).

2. SecurityFocus: sniffers, http://www.securityfocus.com/templates/tools_category.html?category=4&platform=-&path=|%20sniffers%20|

3. S. Sipes, "Why Your Switched Network Isn't Secure," http://www.sans.org/newlook/resources/IDFAQ/switched_network.htm, The SANS Institute, September 10, 2000.

4. S. Wallström, B. Lincoln, IMHO Webmail, *http://www.lysator.liu.se/~stewa/IMHO*.

5. G. Cohen, B. Knight, SecureFTP, *http://secureftp.sdsc.edu*, 2000.

6. Y. Last, WebRFM, http://www.geocities.com/SiliconValley/Horizon/7772/webrfm.html, 1999.

7. "Kerberos, the Network Authentication Protocol," *http://web.mit.edu/kerberos/www/*, September 10, 2000.

8. "OpenSSH for Windows and Mac", *http://www.openssh.org/windows.html*, July 25, 2001.

9. M. Crispin, RFC 2060, "Internet Message Access Protocol – Version 4rev1," December 1996; J. Myers, M. Rose, RFC 1939, "Post Office Protocol – Version 3," May 1996; A. Freier, P. Karlton, P. Kocher, "The SSL Protocol Version 3.0," *http://home.netscape.com/eng/ssl3/draft302.txt*, November 18, 1996.

10. R. Kaseguma, sslwrap, *http://www.rickk.com/sslwrap/*, 1999.

11. Protocol Numbers and Assignment Services, *http://www.iana.org/numbers.html*, Internet Assigned Numbers Authority, April 30, 2001.

12. qpopper, *http://www.eudora.com/qpopper/*.

13. "Email FAQ," *http://www.netgate.net/html/email_faq.html*; "Changing POP (or other) Port in Eudora," *http://www.eudora.com/techsupport/kb/1501hq.html.*

14. J. Myers, RFC 2554, "SMTP Service Extension for Authentication," March 1999.

15. Sendmail, *http://www.sendmail.org*.

16. J. Myers, RFC 2222, "Simple Authentication and Security Layer (SASL)," October 1997.

17. EGD, *http://www.lothar.com/tech/crypto/*.

Windows password encryption is known to be weak.[23] We only use this protocol on internal networks and limit it to Windows-only networks.

We are also vulnerable to keystroke sniffers. An intruder at a remote site can install a keystroke sniffer and intercept passwords as they are typed. In our experience, keystroke sniffers are rather rare, and our risk is relatively low. The only solution to this would be one-time passwords or hardware tokens, which for us is not worth the expense.

Policy

It probably goes without saying that it is important to have policy behind your technology.

As always, support from management is instrumental. Our management takes security seriously and has supported our efforts. What has helped us gain that support is showing that we are enabling services, as opposed to denying access. Additionally, we make sure that management knows what we are doing, and is comfortable using the technology, before switching off services. In this way, when the disgruntled big-ego researcher calls the director to complain about not being able to use Telnet, the response is, "I'm able to use it, why can't you?"

We sometimes appeal to ego to encourage recalcitrant users. For instance, when dealing with a researcher, first we'll find out who their biggest rival is. The conversation then goes something like this: "Well, Dr. X, Dr. Y [the rival] had no problem installing and using SSH. Perhaps we could ask one of his grad students to come over and show you how to use it." This works more often than you might think.

The other key strategy is to give users plenty of advance warning. We typically give six months' to a year's warning, with email reminders, items in the message-of-the-day, and banners on services. Even with all this notice, some users will not get the information. So we make sure that help-desk staff are prepared to support them. In some cases, we preemptively help out users who we know will have difficulty.

Clients that store plaintext passwords for the convenience of the user are problematic. An intruder on a remote machine can pluck these passwords out of the files where they are stored. We don't have any (technological) way to prevent users from using these features. We do ban user storage of plaintext passwords on our managed systems.

Gotchas and Issues

When configuring any encrypted service, first make sure that authentication works properly without encryption. It can be easy to assume that there is a problem with session encryption when the real problem is that authentication is failing.

When enabling encryption, verify that the transmitted passwords are actually encrypted. Some services can be easily misconfigured, so that you think that passwords are encrypted when they actually are being sent in plaintext.

Password distribution/management is not trivial. All of these systems require that we manage user passwords on a variety of systems. We do not use a centralized account management service (e.g., NIS), because most of them are insecure and/or don't work with all of our systems.

We have a home-grown Web-based password changing system, which sets UNIX passwords, Kerberos pass-phrases, APOP passwords, and Windows passwords. We do not

manage all passwords from a centralized database but explode the passwords out to their respective systems. It's not the best system, but it works for us.

Be aware of software that people can install on their desktops on their own, such as VNC (it can be SSH-wrapped), personal Web servers, FTP servers, etc. We ban these as a matter of policy, but it is difficult to prevent.

Future Directions

We eventually will replace /bin/login with the Kerberized version. This version will log users in using their Kerberos pass-phrase, and get a ticket-granting ticket all in one shot. This will allow us to use the KDC as the central password management system for UNIX logins. We would like to integrate Windows 2000 into this environment, but its feasibility remains to be seen.

We are evaluating a Web-based system for users to access our file systems (WebRFM, see note 6), providing the ability to upload and download files through an encrypted, authenticated site. The system looks promising.

We have also started looking at OpenAFS[24] with Kerberos. AFS provides true user-authenticated file sharing and can be used effectively for file sharing between different sites.

Conclusion

Due to the ubiquitous use of sniffers, disabling plaintext passwords is critical for effective protection of systems. There is no single solution that provides universal access, but effective service can be provided through a combination of technologies, policy, and careful user handling.

Thanks to the following people who were involved in the actual implementation: Tom Guptil, Tom Perrine, Jeff Makey, Cindy Zheng, Haisong Cai, and Dave Savilonis.

For additional documentation see *http://security.sdsc.edu/self-help/no-plaintext/*.

18. B. Bannister, "Implementing Authenticated SMTP with Sendmail," *http://security.sdsc.edu/publications/smtp-auth.shtml*.

19. "Port Forwarding," *http://www.ssh.com/products/ssh/administrator30/Port_Forwarding.html*, May 2001.

20. Roxen Web Server, *http://www.roxen.com/products/webserver/*.

21. Netatalk, *http://netatalk.sourceforge.net/*.

22. Pretty Good Privacy, *http://web.mit.edu/network/pgp.html*.

23. l0phtCrack, *http://www.atstake.com/research/lc3*.

24. OpenAFS, *http://www.openafs.org/*.