Using SecurID tokens in an open multi-host UNIX environment

Wietse Venema
Mathematics and Computing Science
Eindhoven University of Technology
The Netherlands
wietse@wzv.win.tue.nl

1. Product evaluated and product evaluation platform.

The product evaluated was the Security Dynamics Inc. ACM/8100 version 1.0 host software for SunOS 4.1.1b or later, and a small number of SecurID tokens (the credit-card-sized type without keypad). The evaluation was done with Sparcstations SLC and ELC running SunOS 4.1.1.

The same product is available for other environments. The findings discussed in this report will be valid for most other UNIX environments.

2. The challenge.

The purpose of this review was to study the use of SecurID authentication tokens in an open university computing environment. Such an environment typically comes with a multitude of UNIX workstations, X-based window systems, and file sharing with NFS. The typical UNIX environment provides some 20 ways to obtain access.

The motivation was to see how much functionality would have to be sacrificed in order to make the use of SecurID tokens practical, assuming that no modifications would be made to existing software.

3. Organization of this report.

After a summary of the operation of the SecurID tokens and ACM (Access Control Module) host software, their impact on UNIX system access methods will be discussed. The access methods are grouped according to access type (terminal login, remote command execution, remote file access etc.). With each access method, it will be discussed whether or not such access is possible or desirable in a SecurID-protected environment.

4. Summary of operation.

Operation of the ACM host software is elegant and simple. The user´s login shell field in the password file is replaced by the pathname of the *sdshell* program. After the user has logged in in the usual manner, the *sdshell* prompts the user for a passcode. The user has to enter a secret PIN code and the six to eight digit pseudo-random number that is displayed by the user´s personal token, and that changes every minute. According to the documentation, a passcode (PIN plus pseudo-random number) can be used to login only once (but see below).

After entry of the correct correct passcode, the *sdshell* program executes the interactive UNIX shell program that is specified in the ACM database. The *sdshell* program also sets the *SHELL* environment variable to the pathname of the user´s interactive shell, so that there will be no passcode challenge when the user spawns a new shell processes in the course of the same login session.

This approach is elegant because it involves no change to existing system software. It also has limitations: the *sdshell* program does not directly support access methods other than terminal logins. Another limitation is that the evaluated software did not offer a programmatic interface.

5. Login access methods.

5.1. Terminal logins.

Terminal logins via hard-wired lines, via modems or via network services such as *telnet* or *rlogin* ultimately go through the *login* program. As long as the latter program invokes the ACM *sdshell* program, the only way to get through is to provide the correct passcode.

Once a passcode has been used to access a specific account, it cannot be used again to access that account. Thus, the *sdshell* provides protection against replay attacks by eavesdroppers.

However, the protection against replay attacks is less effective when the same token can give access to more than one host. When an eavesdropper has captured a user´s passcode, that passcode can still be used to access the other systems that the token gives access to. This can be a problem in an environment that consists of multiple hosts on an untrusted network.

5.2. su

The *su* command is an alternative way to obtain a user´s privileges. Ordinary users must specify the target account´s password in order to switch privileges (unless the target account has no password). The super user can switch privileges without having to specify the password.

When the target account is protected by the *sdshell* program, what will happen depends on who is making the switch. If it is an unprivileged user, she will be prompted for a passcode. However, when the super user makes the switch to a protected account, no passcode is required. This test of who makes the switch requires some knowledge of kernel implementation details but it can be done in a reliable manner.

Conclusion: the interaction between the ACM software and the *su* command is adequate.

5.3. X-based login tools.

The X Window System[1] comes with the *xdm* graphical login tool. Some vendors provide their own alternative, but basically these programs all perform the same task: display a nice logo on the screen and prompt for a login name and a password. The login tool checks this information against the UNIX password database. When the correct password

1. The author´s observations are based on the MIT X Window System version 11, release 5.

is given an X session is started: either a system-provided default desktop, or a desktop according to the commands in a user-provided *.xsession* file in the user´s home directory.

I noticed several problems with the combination of SecurID and X-based login tools:

- When the default desktop is used, the user will be prompted for a passcode each time an interactive shell window is created. Because of the protection against replay attacks, the user will have to wait until the token has generated a new pseudo-random number (once per minute).
- The user can provide a customized *.xsession* file that specifies a suitable *SHELL* environment variable, and avoid the passcode challenge altogether.

Thus, the passcode challenge must be implemented *before* the desktop is started. This requires modification to the *xdm* login sequence. The *sdshell* program, intended for text-based logins, is not directly suitable for this task. It is conceivable that the graphical login tool is modified so that it prompts the user for a passcode, runs the *sdshell* program from a pseudo terminal, feeds it the user´s passcode, and examines the program´s response, but it would be desirable that the ACM host software provides a suitable programmatic interface.

Conclusion: X-based login tools require customization to enforce passcode validation.

5.4. Screenlock programs.

The purpose of screenlock programs is to take over the keyboard and display until the correct password is given. Such programs come with almost every windowing system and are even available for text-based terminal environments. These programs must be modified to make use of the ACM software. Again, a problem is the lack of a suitable programmatic interface to the ACM host software.

6. Remote file access.

6.1. NFS

The Network File System, originally developed by Sun Microsystems, provides access to files that reside on other hosts. Access is handled at the operating system level and is therefore transparent to most applications. NFS relies on an RPC (Remote Procedure Call) transport layer that permits different flavors of authentication:

- The default is to leave all authentication up to the NFS client host. All the user has to do is to ‘prove´ her identity to the NFS client. As long as the NFS server is willing to provide service to the client host, the server simply believes the numeric user and group ID values that the client host sends. NFS security can be improved somewhat by refusing requests that come from an unprivileged port[2].
- Later NFS implementations (e.g. SunOS 4) can be configured to use so-called *secure* RPC. In this mode of operation, the identities of users, client hosts, and server hosts, are established by cryptographic means (DES and public/private keys).

---

2. A UNIX-specific convention: network ports with numbers below 1024 are called *privileged* because they can be allocated by the superuser only. Ordinary users can allocate only *unprivileged* network ports (ports in the range 1024..65535).

When authentication is left up to the NFS client hosts, anyone who can achieve superuser privileges can generate arbitrary NFS requests on behalf of arbitrary users and hosts. Thus, client-based authentication is sufficient when there are no untrusted hosts on the same networks as the clients and the servers.

With secure RPC it becomes much more difficult to impersonate users or hosts; especially when the time-to-live of the authenticators is reduced from the default 3600 seconds to a shorter period of a few minutes.

Conclusion: With the present ACM software, where the use of tokens is limited to terminal logins, the tokens cannot be used to generate keys for secure RPC. Thus, NFS may be the weakest link.

6.2. pcnfsd

The *pcnfsd* program provides some elementary services for NFS client software that is used on, among others, MS-DOS personal computers. One of the services is "authentication": the PC prompts the user for a login name and password, sends it to a *pcnfsd* process on a server, and awaits an YES/NO response. When the answer is positive, the PC software will accompany all subsequent NFS requests with that user´s uid and gid numeric values.

Older *pcnfsd* versions will not detect that the user has a non-standard login shell. Thus, the user can access her files without passcode challenge. Newer versions of the *pcnfsd* program will refuse to authenticate users with restricted shells such as the *sdshell* program.

The *pcnfsd* authentication process is easily worked around by a knowledgeable user who knows how to patch disk files or memory. Thus, in an environment where accountability is important, the use of PC-based NFS products with client-based authentication is to be discouraged, at least for access to sensitive files.

6.3. rcp, rdist

These file distribution protocols rely on the *rsh* (remote shell) service (discussed below), and thus do not support passcode validation. They have the same problems with respect to accountability as the underlying *rsh* service and should not be permitted on systems where accountability is important.

6.4. ftp

The *ftp* file transfer protocol involves the exchange of a user name and a password, but there is no provision to specify additional information such as a SecurID passcode.

Most UNIX ftp server implementations (including SunOS 4) have the notion of a restricted shell: access is denied when the user´s login shell is not specified in the file */etc/shells*; when that file does not exist, access is denied when the user´s login shell is not one of the standard UNIX shell programs. An alternative is provide a file */etc/ftpusers* that lists all users that should be denied *ftp* access.

Thus, on systems that have a notion of restricted shells, accounts that are protected by the *sdshell* program automatically lose *ftp* access. On systems that do not have the notion of restricted shells, the ftp service should be disabled.

7. Remote command execution

7.1. on/rexd

A facility that is often found on systems with NFS is the *on* client program, and its companion daemon, *rexd*. This combo provides a more user-friendly form of the *rsh* or *rlogin* services: to the extent that it is possible the full environment, including working directory and environment variables, is made available on the remote system.

The typical *on* implementation uses the same RPC transport layer as NFS. Again, the default is to leave all authentication up to the client software. However, unlike NFS, the typical *rexd* implementation has no provision for access control[3] and does not require the use of a privileged port. Consequently, the default *rexd* provides no protection whatsoever against impersonation. For this reason the *rexd* daemon is usually disabled.

Even the secure RPC version of *on/rexd* presents a conflict with the use of SecurID tokens. The client can specify what program is to be executed on the remote host, so that it is trivial to bypass the *sdshell* program.

Recommendation: disable the *rexd* service because it can be used to avoid the passcode challenge.

7.2. rsh

The *rsh* (remote shell) protocol provides a simple form of remote command execution. Authentication is minimal, though it is better than with *on*: requests from unprivileged ports or from untrusted hosts are rejected.

The *rsh* protocol has no provision for password exchange. The server believes the remote user identity that the client host provides in the form of a clear-text string. Such information is reliable provided that the client host and the network are not compromised.

I was somewhat surprised to find that the *sdshell* program is entirely transparent to users of the *rsh* service[4]. Unfortunately, the transparency for *rsh* execution is not restricted to the owner of an account: the *sdshell* program does not know the identity of the remote user. Thus, unprivileged users can set up private *.rhosts* files to grant other users access to their accounts without passcode challenge. Another example: when a trusted host is compromised, an intruder can use *rsh* to get full access to accounts on other systems that trust the compromised host.

Recommendation: disable the *rsh* service because it can be used to circumvent the passcode challenge.

---

3. Some DG/UX *rexd* versions require that the client knows the name of a host in the server´s */etc/hosts.equiv* file. The author has found that SunOS 4, SCO UNIX 3.2 and others do not have this requirement.

4. Actually, the way this transparency is achieved depends on kernel implementation details. Under SunOS 4.1.2 and later, the SunOS 4.1.1 *sdshell* program is unable to detect that it is run in the context of an *rsh* command, and execution fails silently.

7.3. rexec

The *rexec* protocol provides yet another remote command execution service. The protocol is pretty simple: the client sends an optional port number for the server´s standard error stream, a login name, a clear-text password, and the command to be executed. The command is given to an instance of the user´s login shell, and the remainder of the connection is devoted to moving data between the client and the server processes. There is no provision to specify an additional password such as a SecurID passcode.

Knowing that the *sdshell* program will grant access without passcode challenge when run as a child of an *rsh* daemon process, I was not surprised that the *sdshell* program could also be convinced to grant shell access without passcode challenge when run as a child of an *rexec* daemon process.

Recommendation: disable the *rexec* service because it can be used to work around the passcode challenge.

8. Email

8.1. Commands in .forward files.

Many UNIX systems, including SunOS 4, come with the *sendmail* mail transport agent. This program permits the forwarding of messages, not only to accounts or hosts, but also to files or programs. In addition to the system-wide *alias* data base there is provision for an individual user to maintain a personal *.forward* file in her home directory. Commands in a *.forward* file will be executed with the privileges of the owner of that file.

Obviously, the user is not going to be around every time when mail arrives. This is not a problem: commands in the *.forward* file are executed by one of the standard system shells, and not by the user´s login shell.

Conclusion: use of *.forward* files is transparent to SecurID users.

8.2. pop

The Post Office Protocol provides password-protected remote access to a user´s personal mailbox, whether or not that user has a restricted login shell such as the *sdshell* program. For this reason, it may be desirable to disable the *pop* service.

9. Unattended command execution.

9.1. cron

The *cron* daemon is used for *periodical* unattended command execution. Some UNIX systems permit unprivileged users to have personal *crontab* files (including SunOS 4). All cron implementations that are known to the author either invoke a standard shell program for the execution of commands (including SunOS 4), or permit the user to specify the shell program that cron should use.

Use of the *cron* service is transparent to SecurID users.

9.2. at

The *at* facility is used for *one-time* unattended command execution at a specified time. It is the closest approximation to batch processing that can be found on a typical UNIX system. All *at* implementations known to the author either invoke a standard shell program to execute the commands (typical for System V), or rely on the *SHELL* environment variable (including SunOS 4).

Use of the *at* service is transparent to SecurID users.

10. Conclusions.

The ACM/SecurID kit provides excellent authentication provided that the *rexecd*, *rexd* and *rshd* remote command execution services are disabled, and that the *nfs* service is used with care. The use of tokens is attractive because it eliminates the problem of password selection and of password reuse.

A programmatic interface would be desirable so that passcode challenges can be implemented within screen lock and graphical login tools.

The use of tokens alone does not solve the problem of authentication in a distributed environment. This problem is tackled by schemes such as secure RPC and Kerberos that provide strong authentication without any password ever going across the network. I would be most interested in a combination of token-generated passwords that can be used for the generation of secure network authenticators.

service	passcode?	comments
login	yes	
telnet	yes	
rlogin	yes	
su	yes	by ordinary user
su	no	by privileged user
xdm	no	replace or disable
screenlock	no	replace or disable
nfs	no	can be the weakest link
pcnfsd	no	"old" version, replace or disable
pcnfsd	fail	"new" version, sdshell is restricted
rcp	fail	uses rsh
rdist	fail	uses rsh
ftp	fail	sdshell is restricted shell
tftp	no	disable
on/rexd	no	disable
rsh	no	disable
rexec	no	disable
.forward	no	uses /bin/sh
pop	no	disable
cron	no	uses /bin/sh or $SHELL
at	no	uses /bin/sh or $SHELL

11. Acknowledgements.

The tokens and software were kindly made available by Unisource Business Networks Nederland b.v., in cooperation with SURFnet b.v..