Scripting as a Method of Establishing a Reliable Baseline Posture
George Montcrief
November 20, 2000

Sites that employ many computers, relative to the number of people available to maintain them, must find cost- and labor-effective ways to reliably control the configuration of those machines. A significant level of effort, cost and time is required to install, configure and verify the configuration of a computer, and computers that will be connected to the Internet require an even greater level of effort if they are to be reliable and available. Hill [1] points out that "persistence pays off" over time when trying to adequately secure computing resources against attackers, because most organizations don't have the resources necessary to immediately make all the necessary changes.

A short list of the necessary changes might include adoption of security goals, upgrades to operating systems and other software, application of patches, evaluation and installation of access control lists for network equipment and firewalls, user configuration particulars, backups, and evaluation and review of the changes made against the goals of security. A very large part of the work to adequately handle security incidents is done *before* the incident ever occurs, so that recovery is *possible* [2].

One way to make this large amount of work more manageable is to establish and document a denominator of steps for a given platform, and automate those steps. Make a checklist of the configuration steps that must always be performed, and find a way to automate those steps with minimum effort. Every organization will find that they have expectations of minimum changes to the default configuration, and these can be documented using checklists. The are several benefits to checklists, including:

1. Checklists make sure important changes aren't forgotten.
2. Checklists are easily examined for modifications or improvements.
3. Checklists improve manageability of changes through the organization of steps.
4. Checklists remove the dependence of making all changes from your memory (your checklist will still work if you have a bad hair day).
5. Checklists provide a minimum level of reliability of changes.
6. Checklists encourage customization to an organization's particular requirements.

As part of a defensive posture, checklists aren't as sexy as intrusion detection systems costing tens of thousands of dollars, or as high profile as firewalls as a first line of defense against external malicious activity. Though intrusion detection systems and firewalls may be the first line of defense that an administrator wields against an attacker, the best defense is preparation. Checklists, though sometimes low-tech, may be a very effective way of preparing.

The SANS Institute has published a list of Essential Security Action items [3] as a Step-By-Step guide to secure computing. This list consists of guidelines, organized into three levels of priority, that organizations should seriously consider implementing. These actions are organized as a checklist.

The SANS Institute also publishes a series of very useful checklists, known as the Step-By-Stepã guides, to provide specific guidance for administrators of Windows NTâ , Solarisâ and Linux. These include the combined expertise of a large number of practitioners and provide explanations regarding why an administrator might want to make the suggested changes, as well as the how-to of making the changes. In virtually any set of changes to a default system configuration, some of those changes will always be performed every time a system of that type is installed. For example, many vendors ship their operating systems configured to include passwords in the password file by default, and require execution of another program besides the 'passwd' program to store encrypted user passwords in the shadow password file. A script or other executable program may very easily perform these steps, making it unnecessary for an administrator to do so interactively.

Consider, for example, a checklist of the following items:

1. lock all non-root accounts
2. edit /etc/syslog.conf and add a line directing copies of all syslog entries to the site loghost
3. copy the login warning banner to /etc/motd and /etc/issue
4. mount shared filesystems
5. remove or comment unneeded services from /etc/inetd.conf
6. install Network Time Protocol daemon, and copy configuration file from configuration server
7. etc…
8. reboot for all changes to take effect
9. after reboot, check to make sure that changes actually took effect

Strictly speaking, checklists include a box to draw a check in, signifying that an action has been taken, but writing a simple script using any of the three or four most popular login shells or Perl may accomplish all of the above items easily, quickly, and most importantly, reliably. See Appendix A as an example of a short Bourne shell script to automate the first six steps above.

At least one such free, publicly-available, automated open source method of making changes to Linux exists, called Bastille-Linuxã . The root of the word "Bastille" means fortress, or fortified; Bastille-Linux is a series of Perl scripts written by a small group of knowledgeable individuals to help the less-experienced, or more harried, administrator fortify their Linux operating system against many obvious attacks in an expedient, intelligent fashion. Bastille-Linux has several significant values [4]:

1. "Bastille can now run on Non-Virgin systems," meaning that even if the system is not in a default configuration, Bastille will still be able to modify the system without breaking it

2. "Bastille can now run multiple times," meaning that it may be run more than once for multiple modifications

3. "Undo functionality," meaning that the administrator can back out of changes made if necessary without removing and re-installing software

4. "Log-only functionality," meaning that Bastille will tell the administrator what changes it would make if given the opportunity, making the administrator more knowledgeable about the system

5. "Easy extensibility to other distributions," not limiting the administrator to changing RedHat® , SuSE® , or any other distribution, both because it is written to be increasingly Linux-platform independent, and because it is written in Perl.

Point 4, above, invites a few more comments. Bastille-Linux, as an automated method of establishing a security baseline, is especially valuable because it provides explanation to the administrator regarding both the change and the rationale for the change. The "Log-only functionality" provides a log file for the administrator to read, describing the changes that would have been made, providing both a record of changes made and honing the administrator's skills and understanding.

The changes that Bastille-Linux will make for the administrator include:

1. enabling and configuring IP chains
2. downloading and installing updates to installed RPMs

3. setting restrictive permissions on system utilities preventing normal users from executing the utilities
4. adding a second administrator account to allow tracking of the default administrator account
5. disabling remote services such as rlogin and rsh
6. enabling password aging
7. creation of user accounts
8. restricting use of cron to administrators
9. password-protecting the LILO prompt
10. reducing the LILO prompt time to zero seconds to prevent boot options from being used to control the boot sequence
11. disabling CTRL-ALT-DEL reboot
12. password-protecting single user mode
13. editing /etc/inetd.conf and /etc/hosts.{allow|deny} to optimize the use of TCP wrappers
14. adding "Authorized Use" warning banners to /etc/motd and /etc/issue
15. disabling the compilers
16. limiting system resource usage
17. restricting console access
18. enabling additional system logging
19. enabling process accounting
20. minimizing use of privileged daemons
21. installation and use of Secure Shell
22. deactivating the following of symbolic links for webservers
23.disabling of printing using lpd

Though some of the above changes require more than one yes or no answer, they all can be accomplished through yes or no answers. They are also a good example of the changes possible by automating routine configuration checklists. In the case of Bastille-Linux, the automation could be carried a step further by deciding which of the above changes should be made every time a computer is configured, and running those Perl scripts with the appropriate arguments. Running Bastille-Linux takes a few minutes, but is much faster than doing all the same things by hand.

An example screen from Bastille-Linux has been added as Appendix B.

One further important item should be included in any checklist written to securely configure an operating system. Always check behind the changes to make sure that they have taken effect as expected. A change by a vendor to the provided system software may prevent a checklist item from making the expected modification, thus invalidating the baseline security posture. This check may also be scripted to provide the maximum savings of time and effort, and may be as simple as examining the process table listing to determine whether or not telnetd is using the proper arguments.

There is no substitute for thinking through your security posture, and making the mundane dependable. Scripting checklists can do this reliably.

References

[1] Hill, Siegfried. "Promoting Security from the Middle." Information Security Reading Room. V2.73. 12 November 2000. http://www.sans.org/infosecFAQ/sec_middle.htm (18 November 2000).

[2] The SANS Institute. "Computer Security Incident Handling: Step-By-Step." Copyright 1999.

[3] The SANS Institute. "Essential Security Actions: Step by Step." Copyright 1999. http://www.sans.org/newlook/resources/esa.htm (18 November 2000).

[4] Beale, Jay and Lasser, John. "Bastille-Linux." 7 June 2000. http://www.bastille-linux.org/version-1.1.html (18 November 2000).

[5] Garfinkel, Simson, and Spafford, Gene. "Appendix A: Unix Security Checklist." Practical Unix & Internet Security, 2nd Edition. April 1996.

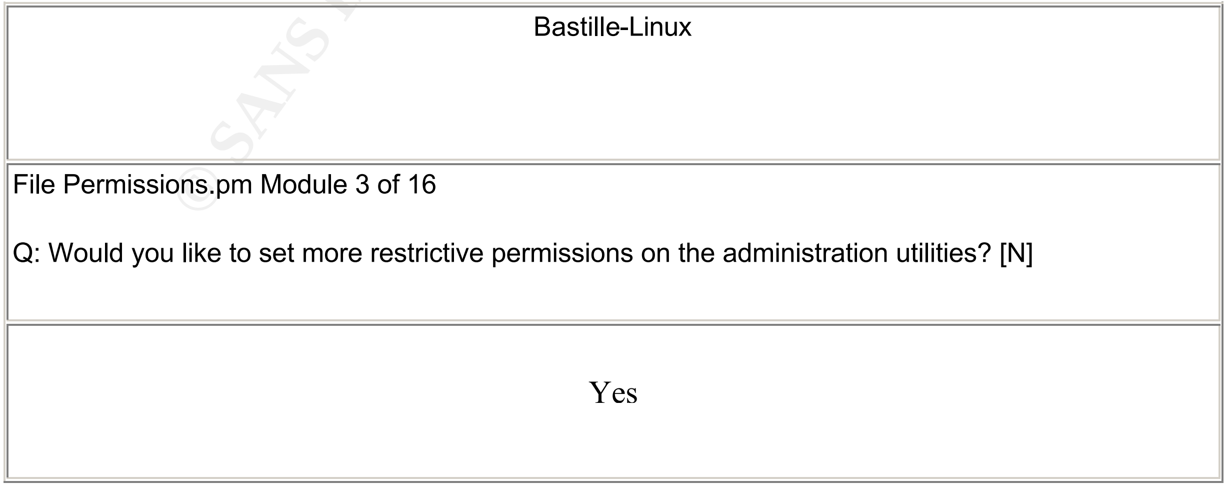
Appendix A

Figure 1. Sample configuration script

```
#!/bin/sh
# lock all default user accounts
for USERACCT in sysadm cmwlogin diag daemon bin uucp sys adm lp nuucp auditor;
do
     /usr/bin/passwd -l $USERACCT
done

# Add loghost to copy syslogs
echo "*.debug       @loghost.domain.com" >> /etc/syslog.conf

# Copy login warning banners to /etc/motd and /etc/issue
/usr/bin/scp securehost:/config/motd /etc/motd
/usr/bin/scp securehost:/config/issue /etc/issue

# mount the site-shared filesystems
echo "securehost:/share1  /remoteshare  nfs3,rw,nosuid,bg  0 0" >>  /etc/fstab
/sbin/mount -v /remoteshare

# remove unneeded services in /etc/inetd.conf
/sbin/cp /etc/inetd.conf /etc/inetd.conf.vendor  # save an original copy
cat /etc/inetd.conf | /sbin/sed /telnet/d        # remove telnetd
cat /etc/inetd.conf | /sbin/sed /login/d         # remove rlogind
cat /etc/inetd.conf | /sbin/sed /shell/d         # remove rshd

# install and start ntp
/usr/sbin/inst -a -f /sharedir/packages/dist/ntp -I ntp
/sbin/chkconfig ntp on
/usr/bin/scp securehost:/config/ntp.conf /etc/ntp.conf
/etc/init.d/ntp start
```

Appendix B

Figure 1. Example screen from Bastille-Linux configuration script



Bastille-Linux

File Permissions.pm Module 3 of 16

Q: Would you like to set more restrictive permissions on the administration utilities? [N]

Yes

No

< Back > < Next > < Explain Less >

TAB to switch Windows Arrow Keys to switch Buttons