;login:
THE MAGAZINE OF USENIX & SAGE
November 2000 • volume 25 • number 7

THEME ISSUE: SECURITY

edited by Rik Farrow

inside:

REPEATABLE SECURITY

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

repeatable security

by David Brumley

David Brumley is the assistant computer security officer for Stanford University and a consultant with Securify, Inc. David also runs the white-hat security site www.theorygroup.com.

<dbrumley@stanford.edu>

Repeatable Process

After a computer security policy is written, the real work begins – implementing it! Implementation is the process of converting a written policy into a set of specific procedures. Implementation requires the translation of policy statements into current technology on current hardware, an often arduous task.

Implementation is difficult, primarily because picking the right technology involves tradeoffs. One product may streamline a business process yet create numerous security risks. Implementors must decide whether the cost of risk mitigation is less or more than the savings incurred by the software.

To make matters more difficult, current products emphasize features over economy of mechanism. Economy of mechanism gives a clear method for a solution, while features tend to cloud which mechanism is appropriate. Balancing the two is difficult. For example, some surveys show that over 300 dialog boxes must be answered to correctly install and configure Microsoft Windows NT 4.0. Because of the sheer number of mechanisms, implementors may have an incomplete understanding of all the tradeoffs being made.

Ultimately, every organization must decide which technologies it supports and which it doesn't. Sadly, many organizations stop there. Each piece of software supported should also have a supported configuration. The reason: Any piece of software may have thousands of switches and dialog boxes that, when configured differently, create radically different solutions. For example, Windows NT 4.0 Workstation out of the box is very different from the same software configured with C2 security.

Instead, organizations rely upon software installed by hand in an ad hoc fashion. Predictably, error occurs. However, there is a better way.

Cloning a computer can be defined as the process of taking an installed system and duplicating the configuration across many hosts in a repeatable and automatic fashion. By definition, cloning is a way of managing the risk of human error. While cloning does not mitigate the risk of incorrect policy implementation, it does assure that the time and thought spent on a correct implementation is not wasted by the introduction of human errors.

In other words, if an implementation of a policy is correct, cloning ensures that each system cloned adheres to exactly the same standards. Cloning takes the traditionally ad hoc method of manual installation and turns it into a repeatable process with repeatable results. Cloning adds a development cycle to workstation installation and management. As a direct consequence, cloning gives the benefits of a true development cycle.

A clonable image is called a source image. There are two main methods for creating a source image. The first is to install a source host exactly as you want and then duplicate it. The second mechanism is to create your own distribution with all the configuration details self-contained.

An example of the first method would be the Norton Ghost product. Norton Ghost can be used to clone WinTel machines. The first step to using Norton Ghost (and products like it) is to install a source host. That source machine is then loaded onto a distribution server. Machines that you wish to clone contact the distribution server and download the image straight to disk.

RPM-based Linux, such as Red Hat, is a good example of creating your own distribution. You choose which RPMs (Red Hat packages) you wish to install and include them in a certain ftp/nfs directory. Then, when a client wishes to use the distribution, it simply FTPs the image to disk.

Often the first mechanism can be recognized because everything except plug-and-play hardware must be the same between source and destination. This is expected, since the system is simply copied over from the source to the destination without any additional drivers. The second method allows for different types of hardware between source and destination, but is not readily available for all platforms.

Time Saved

What are other reasons to clone machines? Cloning saves time. To illustrate, imagine the classic situation where an administrator must install 100 machines. On each machine, the administrator inserts the system CD, boots the computer, then manually answers each dialog question. Even while the administrator is not answering installation questions, he or she cannot stray far from the computer.

The total time it takes the administrator to install the 100 hosts is equal to the amount of time to install one host times 100. In computer-science notation, we would say the task of completing the installations is accomplished in O(N) time. Now it's unusual for a computer administrator to install 100 systems in a single day, so often the time spent goes unnoticed. However, as the saying goes, you can "nickel and dime" your time away. Time really adds up when installing a few today, a few tomorrow, and ten next week.

With cloning, time is saved by decreasing the total time spent when the number of hosts is increased. This concept is easiest to understand graphically. In Figure 1, the dotted line is the time it takes to manually install computers. The solid line is the time it takes to clone computers. Installing only one system takes less time than cloning, simply because there is a small cost associated with setting up the cloning mechanism. However, the mechanism needs to be set up only once. After installing only a few systems, this ramp-up cost becomes negligible, and cloning becomes profitable.

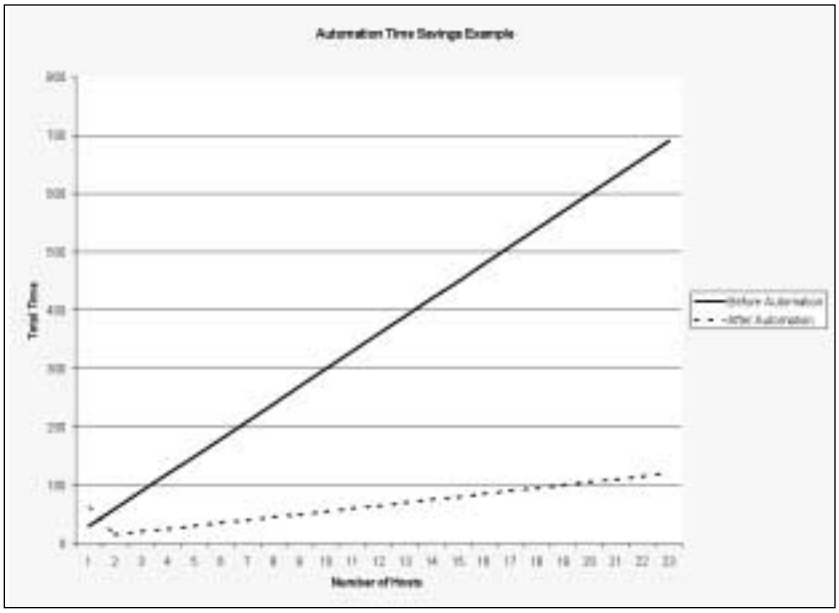


Figure 1

Normally, cloning involves an entire operating system plus any necessary applications. Which operating systems can be cloned? Almost every modern OS has some sort of built-in cloning capability. Windows NT/2000 has the Remote Installation Service (RIS). RPM-based Linux systems allow easy creation of custom distributions. IRIX has a tool named RoboInst. Solaris has JumpStart. The list goes on and on.

Regardless of the specific cloning mechanism, the greater the attention paid to planning your source image (that which you clone from), the greater the benefits. However, the things that make or break a project are how critically you think about incorporating secure authentication, remote administration capability, system security, and productivity applications into your distribution.

Time saved may not be the only advantage of cloning; you may save your organization from a lawsuit!

Authentication and Administration

Providing secure authentication is a necessity in today's hostile Internet. To manage risk, employers must not only ensure that employees' passwords are safe, but must also provide an audit trail of authentication events. Quite often, large lawsuits have been avoided because companies can show that both proper and appropriate action was taken, which can be proven with good authentication data. Time saved may not be the only advantage of cloning; you may save your organization from a lawsuit!

Remote administration allows for automatic updates and troubleshooting of a machine. Typical examples are sudo, Kerberos, and PC Anywhere. To maximize benefits, the remote administration mechanism should leverage off a single sign-on infrastructure. For example, at Stanford University we install Kerberos on each public cluster machine. Kerberos provides encrypted authentication. Kerberos also provides a mechanism for listing principals that can log into an account. We set up our public cluster machines to allow the Kerberos principal "dbrumley.root'" to log in to the root account.

In other words, I can use my active authentication credentials for authorization into various accounts. (This demonstrates a very good reason for distinguishing between authorization and authentication.) Since Kerberos is single sign-on, I can script administration commands to multiple machines. For example, here is a tcsh script to print out the date on each machine:

cat stanford_hosts.list

```
elaine1.stanford.edu
elaine2.stanford.edu
elaine3.stanford.edu
# foreach host (`cat stanford_hosts.list`)
> echo $host
> /usr/kerberos/bin/rsh $host date
> end
elaine1.stanford.edu
Sun Aug 20 10:14:10 PDT 2000
elaine2.stanford.edu
Sun Aug 20 10:14:10 PDT 2000
elaine3.stanford.edu
Sun Aug 20 10:14:11 PDT 2000
```

Notice how the date command above shows it took only one second to execute a command on three machines. More complex scripts can be created, such as mounting a patch tree and installing it. For example:

```
# foreach host (`cat stanford_hosts.list`)
> echo $host
> /usr/kerberos/bin/rsh $host ''mount genesis:/export/home /mnt; cd
/mnt/patches; ./install.sh; umount /mnt;''
> end
```

Stanford uses Kerberos not only for secure single sign-on, but also as a remote administration tool. With Kerberos, a handful of administrators can administer several hundred hosts each with about 30,000 active accounts!

OS Hardening

In their classic book *Firewalls and Internet Security,* Cheswick and Bellovin state as an axiom of computer security that all programs are buggy. A direct corollary is that if you

don't run a program, it doesn't matter if it is buggy. More narrowly, with a UNIX-type system it doesn't matter whether a program is buggy or not if the program never executes with elevated privileges.

Operating System (OS) hardening is primarily concerned with reducing the number of programs that run with elevated privileges, such as network services and setuid programs. A typical hardening script will turn off unneeded services and unused setuid programs to mitigate the risk of exploitation. A side benefit is that if a program is not used, it doesn't need to be patched!

During cloning, it is important that your source image be hardened against attack. All services that are not normally needed should be turned off. A common mistake is to leave services enabled on your source image that are needed only by a small subset of cloned machines. It is much better to disable the services on all systems and manually reenable them when needed. The reason is twofold. First, if you do not need a service on the majority of the systems, you spend more time disabling it on the majority of systems than if you simply enabled it on the few where it was needed. It's just a matter of simple arithmetic. Second, services left on have a tendency to stay on simply because of human error, procrastination, or lack of time.

What if you do not know whether a given service or program needs privileges? One of my axioms of computer security states that if you don't know what it does, you shouldn't be running it. There is a wealth of tools to help you find out what a program does and what it is used for. Instead of simply ignoring the problem, read the man page, ask questions, and do traces of the program before installing it into a source image or production system.

An Example: Creating Your Own Red Hat Distribution

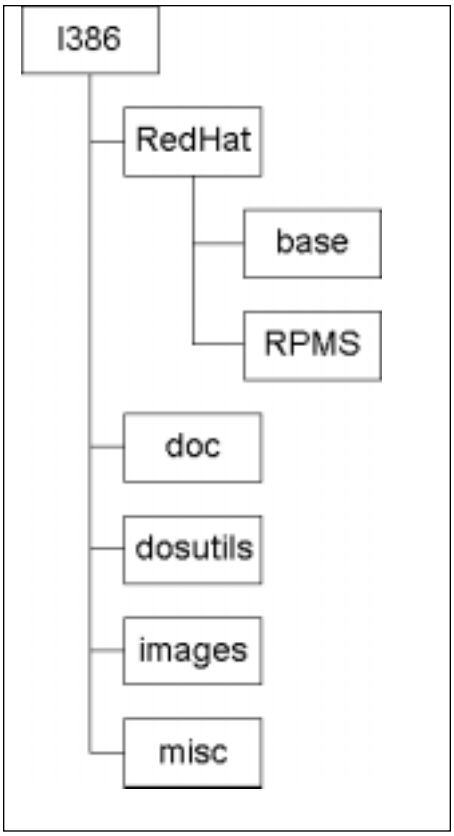
To emphasize how easy it is to make your own distribution, here are the steps needed to create your own Red Hat distribution:

1. Mirror Red Hat.
2. Create your own customization packages.
3. Include your packages in the distribution, remove packages not needed.
4. Inform the installation mechanism of your new package lists.
5. Install away.

Step 1 is to mirror Red Hat Linux. You can either sign up with Red Hat to become an official mirror site, or you can ask one of the primary mirrors if you can mirror off of them. The important thing is to download the directory tree for each version of Red Hat you are going to support.

The i386 directory is the start of the distribution for the x86 architecture. If you like, the same techniques will carry over to the SPARC and PPC directory trees.

Underneath the i386 directory are images, dosutils, Red Hat, doc, and misc. The images directory is where the boot images are kept. dosutils contains programs like rawrite.exe and fips.exe that help users install Linux from a MS Windows system. doc is self-explanatory. misc is an interesting directory, as it contains the source code for the boot and second images. However, there is no need to rebuild the images unless you want to change the verbiage (or something even more drastic) seen during installation. The Red Hat directory contains all the information needed after the initial boot disk to install and configure Red Hat Linux, which we will explore later.



Directory Tree

Step 2 is to create your own packages. There are several books and HOWTOs that describe this process. Ed Bailey's *Maximum RPM* from Red Hat is a good starting place to learn about building your own packages. However, it is a bit outdated, so be sure to consult the online manual pages for possible changes.

There are a few tricks to building successful RPMs for a distribution. The first is that RPMs are installed in a pseudo-alphabetical order. Therefore, if there is an RPM that must run first or last, it's important to name it correctly. For example, my OS-hardening RPM is named "zzsecurity" because it turns off services and disables setuid programs – things I want last during installation to avoid them being overwritten.

Second, I've found it useful to keep custom configuration options in separate RPMs instead of editing the ones bundled by Red Hat. I do this primarily for maintenance reasons; it makes it easy to identify which RPMs I provide and which are part of the standard Red Hat distribution.

Step 3 is to include your RPMs into the standard distribution. This is done by editing the Red Hat components file i386/Red Hat/base/comps. The comps file is a flat text file, with the format:

```
<Component File Format Version>
blank line
<Component 1>
blank line
<Component 2>
blank line
....
EOF
```

If you don't see a component already listed where your RPMs fit, you can create your own component group. The format for each component is:

```
(0|1) (—hide)? <name> {
name1.rpm
name2.rpm
name3.rpm
}
```

Choose either 0 or 1, depending on whether or not you want the package selected by default under custom installation. Also, note that the name of the component is completely arbitrary. For example, Stanford has one called "Stanford," which looks like:

```
1 Stanford {
zzsecurity.i386.rpm
libsafe.i386.rpm
afs.i386.rpm
kerberos.i386.rpm
}
```

If you define your own component, you can use that in later components to make it part of the standard options. For example, to include everything from the "Stanford" component into the "Workstation" component, simply add the name "Stanford" to the workstation component list.

Step 4 is the easiest. When using the Red Hat installer, a database is kept of RPM dependencies, size, and other information. That information is used by the installer to make sure all prerequisites and dependencies are installed properly. After you build

your own RPMs and incorporate them into the component list, that database needs updating. When run from the i386 directory, i386/misc/src/anaconda/utils/genhdlist will rebuild the database for you. Note that the genhdlist command may be different between Red Hat versions, so use the genhdlist included with each version.

Lastly, you should test your distribution. During testing, you are preparing to "go live" with a new environment. Plans for support and maintenance should be in place before deploying sitewide. You'll want to support your Red Hat distribution the same way you would support other software: with a bug repository, a Web page describing basic installation, and so on.

For those who want a working example of a distribution, I've put up all the scripts and RPMs for my distribution at <*http://www.theorygroup.com/Tools/TGLinux*>. TGLinux is based upon a distribution I did for Stanford University that has been successfully installed on several thousand systems.

Branding

When creating a Red Hat distribution, there are several ways to do "lite branding." Here is a short list of ideas:

1. Change the graphical login logo to your own. It's located at <*usr/share/pixmaps/redhat/redhat-transparent.png*>.

2. Incorporate the latest fixes and patches into your distribution nightly. An example script can be found at: <*http://www.theorygroup.com/Tools/TGLinux/scripts/merge.pl*>.

3. Place a common motd and banner in /etc/issue and /etc/issue.net. Note that these files are automatically recreated at boot from /etc/init.d/rc.local normally, so you may have to make some additional changes to that startup script.

4. Burn CD-ROMs of the distribution for home users.

Summary

Although the cloning mechanism may change for other operating systems, certain tenets always apply. First, by creating your own clonable image, you have the opportunity to deploy and enforce your security policy. Second, cloning saves time. Third, cloning lends itself to a true development cycle with all its benefits.

But just as with everything else, the more thought put into planning, the better the results. Too often, we find ourselves typing in the same thing day after day. To have computers do what they should – enhance productivity – anything you find yourself doing multiple times should be automated. By automating tasks, you will create a repeatable process with repeatable results, an utter necessity to compete in the upcoming e-business world and participate in the hostile Internet.

To have computers do what they should – enhance productivity – anything you find yourself doing multiple times should be automated.