Linux Journal Issue #40/August 1997

Features

News & Articles

Reviews

Designing a Safe Network Using Firewalls

Paul Wouters

Issue #40, August 1997

Why you need a firewall and how to best set it up to meet your needs for network security.

A firewall can be your best friend; it can also be the cause of a lot of unforeseen problems. When should you consider placing a firewall into your network? And if you are sure you need one, where in the network do you put it? Too often, firewall policy results from a non-technical board meeting, on the basis of the chairman's “I think we want a firewall to secure our network” remark. An organization needs to think about its reasons for installing a firewall—what is it meant to protect against, and how should it go about doing its job? This article aims to clarify some of the issues that require consideration.

What is a Firewall?

Although this question seems easy to answer, it is not. The security experts say a firewall is a dedicated machine that checks every network packet passing through, and that either drops or rejects certain packets based on rules set by the system administrator. However, today we also encounter firewalls running web servers (Firewall-1 and various NT and Unix machines) and web servers running a firewall application. It is now common practice to call anything that filters network packets a firewall. Thus, the word *firewall* usually refers to the function of filtering packets, or to the software that carries out that function—and less often to the hardware that runs the application.

It is by no means necessary to purchase specialized firewall hardware or even software. A Linux server—running on a $400 386 PC—provides as much protection as most commercial firewalls, with much greater flexibility and easier configuration.

Are Firewalls Necessary?

A few years ago I attended a Dutch Unix Users Group (NLUUG) conference. One of the topics was “Using Firewalls to Secure Your Network”. After listening to a few speakers, I had not found a single argument that justified the necessity of a firewall. I still believe this is basically true. A good network doesn't need a firewall to filter out nonsense; all machines should be able to cope with bad data. However, theory and practice are two different things.

Unless you have incredibly tight control over your network, your users are likely to install a wide variety of software on their workstations, and to use that software in ways you probably haven't anticipated. In addition, new security holes are discovered all the time in common operating systems, and it's very difficult to make sure each machine has the latest version with *all* the bugs fixed.

For both of these reasons, a centrally-controlled firewall is a valuable line of defense. Yes, you should control the software your users install. Yes, you should make sure the security controls on their workstations are as up-to-date as possible. But since you can't rely on this being true all the time, firewalls must always be a consideration and nearly always a reality.

The Ping of Death

A few months ago, a small crisis arose in the Internet security world—the infamous “Ping of Death”. Somewhere in the BSD socket code, there was a check missing on the size of certain fragmented network packets. The result was that after reassembling a fragmented packet, the packet could end up being a few bytes larger than the maximum allowed packet size. Since the code assumed this could never happen, the internal variables were not made larger than this maximum. The result was a very nasty buffer overflow causing arbitrary code to be generated, usually crashing the machine. This bug affected a large community, because it was present in the BSD socket code. Since this code has often been used as a base for new software (and firmware), a wide variety of systems were susceptible to this bug. A list of all operating systems vulnerable to the “Ping of Death” can be found on Mike Bremford's page, located at http://prospect.epresence.com/ping/. A lot of devices other than operating systems were susceptible to this problem—Ethernet switches, routers, modems, printers, hubs and firewalls as well. The busier the machine, the more fatal the buffer overrun would be.

A second reason this bug was so incredibly dangerous was that it was trivial to abuse. The Microsoft Windows operating systems contain an implementation of the ICMP ping program that miscalculates the size of a packet. The maximum packet you can tell it to use is 65527, which is indeed the maximum allowed IP

packet. But this implementation created a data segment of 65527 bytes and then put an IP header on it. Obviously, you end up with a packet that is larger than 65535. So all you had to do was type:

```
ping -l 65527 victim.com
```

Once this method was known, servers were crashing around the world as people happily pinged the globe.

As you can see from the list on Mike's page, patches are not available for all the known vulnerable devices. And even if they were, the system administration staff would need at least a few days to fix all the equipment. This is a situation where a firewall has a very valid role. If a security problem of this magnitude is found, you can disable it at the access point of your network. If you had a firewall at the time, most likely you filtered out all ICMP packets until you had confirmed that your database servers were not vulnerable. Even though not a single one of these machines should have been vulnerable, the truth is that a *lot* of them were.

The conclusion we draw from this experience is that the speed and power of response a firewall gives us can be an invaluable tool.

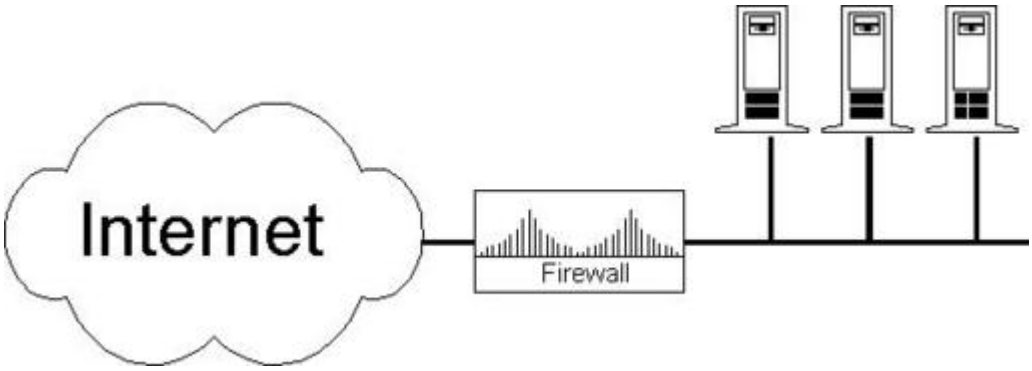
How Do We Secure Our Network with One or More Firewalls?

These are the basic questions you should ask:

1. What do we need to protect?
2. Against whom do we need to protect?
3. Where do we place the firewall(s) in the network?
4. How do we configure the firewall?
5. How do we monitor what is going on?

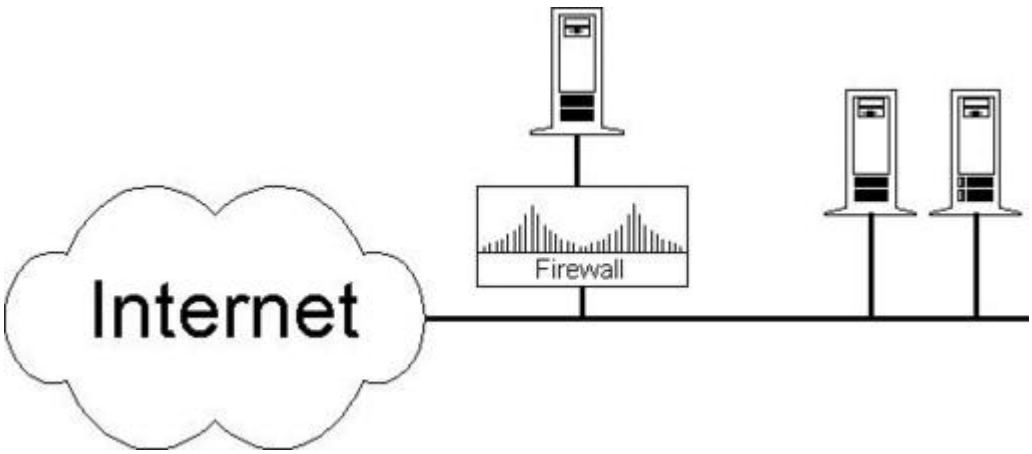
To answer these questions correctly, it is of vital importance that you map your entire network. You don't need to map every single device in the network, since that changes often anyway. Try to map the separate subnets, the routers, the hubs and the physical locations (floors, offices, classrooms). Include the important parts of the network that you wish to secure the most.

What Do We Need to Protect?



Firewall Placement

Most firewalls are used to protect the entire Local Area Network (LAN). In this case, the Internet router usually acts as the firewall. A properly configured Internet router filters out the IP numbers used locally (for instance 10.*, 127.* , 192.x.y.*) to prevent IP spoofing. It should also filter out all packets from the outside with an IP number that normally can come only from the inside. Any packet in this category can only be an attempt to trick your machines, and it should be denied access immediately. Next, filter out any outgoing IP traffic that doesn't have your registered class of IP numbers. This is not only to prevent sending out bogus packets (or to keep your people from spoofing the Internet), it's also for your own security. In particular, Windows products tend to disregard RFCs. One day I found a Windows 95 machine that shared its local printer by giving it the IP number 6.6.6.6. If your Internet router doesn't filter out these packets, you might be routing your printed documents onto the Internet.



Another frequent use for firewalls is to protect a single machine. If you want to protect a single machine with a firewall, you must make sure it doesn't depend on anything outside the firewall; otherwise, your firewall serves no purpose apart from giving a false sense of security. If the protected server is using data from an unprotected PC, someone can falsify the information on the PC in order to do potentially serious damage to your server's data. Someone gaining access to the PC could also reach the server by pretending to be the trusted PC user. If the machine relies on other machines, you want to place your firewall a bit further upstream, so that it can protect those machines as well. NFS is a

good example of an application you would not want to allow through the firewall in this setup. This type of firewall is easy to configure. Block all protocols not in use on the sensitive server, forward only those packets with the server's IP, and don't forget to prevent IP spoofing of your server's IP number from the outside.

A script to set up typical firewall rules to protect a single machine or small subnet is shown in Listing 1. Script for Typical Firewall Rules.

Obviously, real networks aren't as simple as the above examples. Most networks have various machines which are multi-homed and part of different subnets. Larger organizations, like schools, have a problem in that a lot of people have physical access to the Ethernet. The best way to protect portions of these networks is to use subnets on physically separate cables. For example, it would be an excellent policy to give the system administration office a separate subnet, since system administrators often need to use the privileged accounts.

The Complex Network

Quite often a network you wish to protect does only a few limited tasks. On a typical administration network, people want to use the Web, e-mail, POP and quite often a telnet connection to the administrative database server (hidden in a Windows application). Masquerading works best for these networks. It makes sure the individual machines in the administrative network are not reachable (unless the masquerading host itself is compromised, which is next to impossible if it doesn't run any services), while keeping all the basic functionality of being connected to the Internet. This has an additional advantage. Often access to database servers is protected through a TCP wrapper, which allows only a certain set of hosts to access the database. For each new client machine added to the network, an entry into the appropriate /etc/hosts.* files must be made. With masquerading, this entry isn't necessary, since the new machine will be masqueraded and the IP number of the masquerading host is already known to the database server.

If you cannot physically separate the administrative network, you might want to consider using some form of encryption. Kerberos is often used in these cases, but you could also use an ssh-PPP tunnel (ssh is a key-pair encryption algorithm). With ssh you can easily create a virtual private (encrypted) network between your masquerading host(s) and your database server. That should take care of any eavesdropping risks from students booting rogue Linux machines on the network.

With complex networks, it is important to know who the threat is. The threat typically comes from the inside and not the outside, which is protected by the

Internet Router/Firewall machine. Also, don't forget to protect yourself against your modem pool—IP spoofing can occur from there as well.

Configuring the Firewall

There are basically two ways of configuring your firewall. The first and most secure setup is “Deny everything unless we explicitly allow it.” The disadvantage is that you will have a lot of users wondering why certain things don't work. You might consider this approach in a setup where your firewall protects a very small subnet containing only servers and no clients. A script for setting up this type of firewall is shown in Listing 2. “Deny Everything..” Firewall. This type of firewall requires quite a bit of knowledge about how certain protocols work. Do not attempt it unless you have proper documentation and plenty of time to devote to it.

The second, and easier setup is “Allow everything unless we explicitly deny it.” This one makes your network fairly open, but controls a few dangerous or unwanted protocols. For example, some ISPs use this feature to block all “CU-SeeMe” traffic, as this type of traffic can congest their entire network. A script setting up this type of firewall is shown in Listing 3. “Allow Everything...” Firewall.

How Do We Monitor What is Going On?

As you might have noticed in the two previous examples, all deny rules have the option **-o** set to instruct the Linux kernel to explicitly log all denied packets using the syslog facility. If you deny packets without logging them, someday you will be bug hunting for hours before you realize the problem was a packet filter. Depending on how you have configured the syslog daemon (/etc/syslog.conf), these messages will show up in either the /var/log/messages file or the /var/log/syslog file. You should regularly check these log files on your firewall machines. Make sure there is enough disk space to log even an attack that floods you with messages. If possible, use a separate partition for your log files.

Here are a few log entries from our syslog to demonstrate some common problems. These log entries come from our Livingston router, as well as our Linux machine.

```
Jan  2 15:17:57 unreachable.xtdnet.nl 15 deny: UDP from 130.244.101.74.137
to 194.229.18.53.137
```

This is perhaps the most frequent hit our firewall rules get. Port 137 is the NetBios name-service port used by Microsoft Windows machines to look up names in the local network. However, poor implementation and bad configuration often lead to Windows machines making NetBios requests to another machine. These requests might have been generated by a user's telnet,

FTP or even WWW request. You might want to enable your deny rule without the **-o** flag, so that your log file is not filled up with these very common errors. One of our clients had his root partition entirely filled with netbios logs, stopping his genuine logging from operating and almost crashing the server.

```
Jan  2 17:12:34 unreachable.xtdnet.nl  2 deny: TCP from 10.0.3.1.61007 to
194.229.18.29.80 seq 1471CB0, ack 0x0, win 8192, SYN
```

This message is caused by a badly configured host. The IP numbers in the 10.*.*.* range are reserved for local area networks. We found out this host was a misconfigured masquerading host on the Internet, which used its IP number from the local masqueraded network instead of its real Internet IP number. This misconfiguration traveled through many other routers before being caught by our firewall. Large backbone ISPs don't filter out bogus packets, resulting in easy IP spoofing from almost anywhere on the Internet to anywhere else. Don't trust your ISP to filter out anything; do it yourself.

```
Jan 20 06:57:33 unreachable.xtdnet.nl 14 deny: UDP from xx.yy.zz.aa.904
to 194.229.18.27.1112
```

Our firewall proved its value on this attack. Someone tried to ask our RPC daemon (**udp port 111**) which daemons we are running. Even though an attacker can still find most RPC services by doing a total port scan, it is still a good idea not to give them the information so readily. There is almost never a need to have RPC services exchanged with a server on the Internet. Port scans are easily spotted because they leave a giant trace of occurrences of all your filter rules in your log files.

```
Jan  3 22:16:55 unreachable.xtdnet.nl 44 deny: TCP from xx.yy.zz.aa.17231
to 194.229.18.27.23 seq 7A3731D0, ack 0x0, win 49152, SYN
```

This is another true firewall hit. We banned this host after we received a couple of the above attacks on our RPC server. Since the postmaster of this particular site didn't respond, we blocked access for the host on all ports.

```
Feb  4 09:10:17 polly.xtdnet.nl kernel: IP fw-in deny eth1 UDP 0.0.0.0:68
255.255.255.255:67 L=328 S=0x00 I=4 F=0x0000 T=60
```

Port 68 is bootp (DHCP). Some machine was broadcasting and asking for a bootp server. This could be a Win95 computer or even some intelligent HUB which needs an IP number to support SNMP. (This one had us puzzled for months.)

```
Jan 27 09:47:00 masq.xtdnet.nl kernel: IP fw-in deny eth1 TCP 10.0.4.6:1992
204.162.96.21:80 L=48 S=0x00 I=2993 F=0x0040 T=255
```

This machine didn't define the masquerading host as router, so it tried to be smart—but it still didn't find the right gateway.

```
Jan 28 12:23:50 masq.xtdnet.nl kernel: IP fw-in deny eth0 TCP 194.229.18.2:3128
194.229.18.36:2049 L=44 S=0x00 I=23859 F=0x0000 T=63
```

To understand what happened, we need to dig a little into the inner workings of TCP/IP. All connections are identified by a unique combination of source IP, source port, destination IP and destination port. However, to find well-known services, such as telnet, WWW or cache, it is a common practice to use specific ports. To uniquely identify connections to such a well-known service, a random but unique port is allocated on the local machine. If this machine now makes a connection to a well-known service on another machine, it is guaranteed to have a distinguishable TCP/IP connection. Port numbers below 1024 are normally not assigned as random ports, because they're often used or reserved for the well-known services.

Now, let us look back at the log entry. The computer 194.229.18.36 wanted to set up a connection to the machine 194.229.18.2 on port 3128 (the cache server). It first asked the operating system for a unique random port and was given port 2049. It then initiated the connection to the cache server (194.229.18.2 port 3128). The cache server responded by sending its answer back in a packet to 194.229.18.36 port 2049. But 194.229.18.36 is also using firewall rules, and one of its rules is to block all attempts to connect to the NFS service, which, unlike many other well-known services, is not located on a port below 1024, but on port 2049. Thus, the cache server's response is filtered out. You can solve this problem of distinguishing the connection based on whether or not it originated from your site. You can determine the point of origin by whether the SYN or ACK flag in the TCP header is set. The correct way of filtering out connections to port 2049, while still allowing connections to be initiated from it, is as follows:

```
/sbin/ipfwadm -I -i deny -S 0.0.0.0/0 \
 -D 0.0.0.0/0 2049 -P tcp -y -Weth0 -o
```

```
Jan  2 11:22:58 unreachable.xtdnet.nl 38 deny: TCP from 193.78.240.90.8080
to 194.229.18.2.1642 seq F72DA7C6, ack 0xED8FDEA1, win 31744, SYN ACK
```

A similar situation happened here. Port 1642 was assigned by some machine as the random unique port, but the firewall decided that port 1642 was bad. Livingston Portmaster software uses this port to communicate between a Unix host and their routers/firewalls, so that is why we filter these ports for the outside.

In general, try to avoid blocking high ports, and if you do block out a high port, block that port only for the machine that needs the protection. For example, block port 1642 only for your routers and terminal servers, but leave it open for Unix servers. Then, if the router/firewall receives a packet destined for port 1642 on an internal machine, it will pass it to that internal machine even if port 1642 on the router itself is blocked.

A minor drawback is that we are giving away a bit of information to potential hackers. They can check all your IP numbers to see which ones are routers, firewalls or Unix hosts that talk to the routers or firewalls; however, they can usually find that same information through other means as well. For example, the **traceroute** command yields a lot of information about which machines are used for packet transfer and are therefore either a router, a firewall or both. You can also use the **-y** option mentioned in the previous example. Not all hardware routers/firewalls offer these options.

Random Notes

Most attacks on your firewall are simple probing. This is analogous to a person trying your door handle to see if the door is locked. The above firewall rules should protect you against these without much of a problem.

What if a person is trying to find out more than simply whether your door is locked? What if someone appears to have a true interest in *you*? The first sign of this will be a sudden increase in the number of hits on your firewall from a small set of hosts or networks. Your first step should be to contact the system administrator of those systems. If you are really feeling paranoid at this stage, don't e-mail postmaster and don't trust the technical support phone number on their web site. Look up the general number of the company in a paper phone book or dial an operator to assist you. Once you get through to the company, tell them what is going on and offer them as much information as possible. If you can trust the administration of the sites, this usually guarantees that the attacks will stop.

Only rarely does this approach fail—either because the company's administrators are colluding in the attack, or because the attack is coming from a large provider who gives its users access to a Unix shell. For these providers, it is impossible to trace the abuser just from the timestamp of your firewall logs because dozens of people would have been logged on at the time.

So far, this has happened to us only once. We suspected that the administration itself was responsible for the probes and hacking attempts to our sites. We decided to let the hacker *through* our firewall temporarily, so we could gather more information on what they were doing.

We used two tools to gather information. First, we replaced the normal Internet super server (**inetd**) with **xinetd**. This version of **inetd** has the option to log an incredible amount of valuable information. Second, we needed to run a special version of our **nologin** program to make sure the connection stayed up long enough for us to send out an **ident** probe.

```
/* nologin.c */
main() {
  printf("You have no login on this machine.\n");
  sleep(60);
}
```

We enabled the services to be probed in /etc/xinetd.conf. For example to set up the remote login shell **rsh**:

```
service shell
{
   socket_type     = stream
   protocol        = tcp
   wait            = no
   user            = nobody
   server          = /bin/nologin
}
```

And we enabled **ident** lookups and remote host logging for all services:

```
defaults
{
   log_type            = FILE /var/log/xinetd.log
   log_on_success      = HOST USERID
   log_on_failure      = HOST RECORD USERID ATTEMPT
   instances           = 10
}
```

And finally we were ready to open our firewall for these pseudo services on our host.

Be aware that the described level of logging is very high—your log files will be extremely large. But don't be tempted to disable the logging for real services. For example, we logged a few apparently harmless **finger** requests which were followed by probes from another machine on several occasions. The machines responsible for the probes were very uninformative. But this hacker made the mistake of using his normal machine for a **finger** command first, to see if any system administrator was logged on, before he started his probes from the secure system. And his regular machine was running an **ident** daemon, so our logs recorded his user name.

Firewalls Are Needed

As can be seen by the “Ping of Death” example, firewalls can be a life saver. Furthermore, we have seen that it is fairly easy to configure the firewall, once you have some knowledge about how the TCP/IP protocol works.

When visiting one of our clients recently, I peeked at their two firewalls briefly. Both firewalls had an uptime of 108 days. They had been up ever since installation of Alan Cox's ping patched Linux kernel version 2.0.23. One firewall, protecting the main Internet server, logged four attempts to send oversized ping packets. It also prevented access by some students trying to use illegal IP numbers (whether by mistake or on purpose was not known). It also logged various misconfigured machines sending out bogus IP traffic. The firewall that

protects their main Internet server (which also handles a full Usenet news-feed) had routed close to a terrabyte of IP traffic. Their firewalls have proven to be a very stable and valuable addition to their network security, where they have to be concerned about not trusting internal machines as well as external machines and where total control of the Ethernet cables is not guaranteed throughout the entire complex.

Glossary

Paul Wouters started his Unix experience with Linux 0.99pl8, so he could program an MUD at home. He is currently a system administrator for Xtended Internet, where his exposure to a wide variety of Unix flavors has only increased his love for Linux. At work he can be found idling at earthmud.org. He can be e-mailed at paul@xtdnet.nl.

Archive Index Issue Table of Contents

Advanced search

Tripping up Intruders with Tripwire

Kevin Fenzi

Issue #40, August 1997

Tripwire is a program designed to help you discover if someone is tampering with your system.

You run the latest Linux kernel, read the Linux-security-alert mailing list, install security patches as soon as they come out. Your Linux machine is secure. Right?

Alas, it's not very likely. Even if you install the newest patch, someone might have already compromised your machine and replaced your system binaries without your knowing it. In that case, fixing the hole which allowed them entrance is just part of your problem. Even with quick turn around in the Linux community (we have the source code after all) for security vulnerabilities being spotted and someone coming out with a fix for it, there will always be a window of opportunity for someone to compromise your machine.

What more can you do to close this window? How can you know when some intruder has broken it? Enter Tripwire. Tripwire has a pretty simple concept. It takes checksums of all your important files; then later, you can check your files against the Tripwire database and determine if any change or tampering has occurred. The current version of Tripwire (1.2.2) has not been updated since 08/94, so it is a very stable program. There is not much that can go wrong with it.

Some cracker tools have been designed to modify or replace system binaries (**login, telnetd, vmlinuz, ps, ls,** etc) and then make them appear to be the same as before. Usually the methods are crude, consisting of just the modification of the date and size of the file (back to their original values). A casual glance shows all to be fine. Some intruder tools even try to get the modified or replaced binary to match a simple checksum. For instance, change the binary so it will pass a simple test such as:

```
cmp /bin/login /cdrom/untouched/bin/login
```

In order to make sure this doesn't happen, Tripwire is capable of using all sorts of checksumming algorithms. Tripwire comes with: The RSA Data Security, Inc. MD5, MD4 and MD2 Message Digest Algorithms, Snefru (the Xerox Secure Hash Function), SHA (the Secure Hash Algorithm) and Haval code. The authors of Tripwire suggest that MD5 is sufficient for most checksumming, and critical files might also be checked with Snefru. Checking with one checksum like MD5 is pretty good, but imagine the difficulty of creating a binary or file that passes 2 or 3 (or more) different checksums. Few system intruders have the time/ resources to even make the attempt.

Using the Tripwire database, you can check all your critical files for tampering. Now, how do you know if someone has tampered with your Tripwire binary or Tripwire database? After all, if the intruder can modify your Tripwire database, you are back to square one.

Several different methods exist. The easiest, and I think the best for most security conscious Linux users, is to place Tripwire and the Tripwire database on a read-only floppy disk. Since most Linux machines have a floppy drive and few are in use all the time, it's a good match. The Tripwire authors also suggest that if you are very concerned, you can print out the Tripwire database. It's hard to imagine an intruder being able to modify documents printed before they broke in. Other possible schemes include: remote mounting the Tripwire binary and database from another more secure machine read-only, putting it on a write-protected Zip disk, or even getting an old, small hard drive that has been jumpered to hardware enable read-only and put it on that. The idea is to put it on some media that you can make read-only in hardware. It does you no good to place Tripwire where an intruder can mess with it. If the machine you want to check is in a place accessible to many people, keep your Tripwire floppy in a safe location and bring it to the machine only when you want to check your files. You can also NFS mount it from a remote, more secure machine with a floppy.

Okay, you've decided to install it. How do you go about that? Well, grab the latest Tripwire source (Tripwire 1.2 patch level 2). It should compile just fine under most Linux distributions. I had no problems compiling it under a Red Hat 4.0 system out of the box. Read through the README file for a quick overview, and if you want more details, take a look at the PostScript design paper. There is also a Tripwire RPM available, but I would recommend against using it, as you can't specify where it should be installed.

Make the choice of the directory where your Tripwire binary and Tripwire database will reside before you compile. Tripwire hard codes these paths to prevent tampering.

Once Tripwire is compiled, take your machine down to single-user mode. I know, you don't want to lose that many months uptime, but you must make sure you are the only one on the system and no one has a remote connection to your machine that can be used for tampering. If you are particularly concerned, you might consider a clean install of your system before installing Tripwire in order to be sure all your binaries are clean. In single-user mode, mount the media on which to place Tripwire and the Tripwire database. (Mount it read/write this time, so you can copy Tripwire onto it.)

Install Tripwire. Next, determine what binaries are important for Tripwire to check. It is almost useless to check files that change a lot (user files). Tripwire is mainly for your important system binaries. My Tripwire configuration checks: **/etc, /sbin, /bin, /usr/bin, /lib** and **/vmlinuz**; it also checks some files in **/dev**. Even these are going to change more than you might think. If you select too much, your Tripwire reports are going to be too full of hits for you to notice a real break-in.

After you get your **tw.config** file set up for the directories you want to change, it's time to make your new, clean Tripwire database. Type:

```
tripwire -init
```

Relax for a while. It takes Tripwire a fair amount of time to generate all the checksums. Using just the default MD5 checksum on all the directories I mentioned above takes my dual 166MHz Pentium about 5 minutes (faster disks would help).

After Tripwire has finished creating the database, remove your Tripwire media and make it read-only (jumper, write-protect notch, etc). Then, bring your machine back up to full multiuser mode and mount your Tripwire media.

Now, to make sure things are working, try changing a file in one of the directories listed in the Tripwire database. Run Tripwire and make sure it catches it. It will output a very nice, verbose report showing which files have changed and in what way they have changed (modify time, checksum, etc).

I have my Tripwire set up to run each night in a **cron** job and mail the results to me. You can have yours do the same or just run it when you suspect something is wrong. I would recommend you run it pretty often. You never can tell when you might be compromised. Checking a short Tripwire report every day is a small investment in security that can really pay off. You can still rerun Tripwire whenever you think there might be a reason for it. (Keep in mind that an intruder might modify your mail from the **cron** job to make it look like nothing happened.) For that reason, I also run Tripwire periodically when I have nothing else going on (while “on hold” for those long tech support calls, etc).

You will find that as you update packages and config files, your Tripwire database flags more and more files; periodically, you need to take your machine back down to single-user mode, remount your Tripwire media read/write and update your database with a different command:

```
tripwire -interactive
```

Make sure you know why each file is different now. If you are at all unsure, don't update that file in the database; instead, replace it with a fresh copy off your distribution site or CD-ROM and then rerun Tripwire.

It's that simple. You now have a nice detector capable of telling you when someone has been tampering with your files, and the peace of mind that comes with knowing your Linux box is the most secure one on your network.

Kevin Fenzi has been fascinated with Unix security since he saw his first crack program. He currently does programming and consulting on many flavors of Unix, but his Linux machine at home is his favorite. He can be reached via e-mail at kevin@scrye.com.

Archive Index Issue Table of Contents

Advanced search

TCFS: Transparent Cryptographic File System

Ermelindo Mauriello

Issue #40, August 1997

A description of how the TCFS secure file system works and why you should use it.

Current network technology makes it cheap and convenient to share resources over a network. Typically, a computer network consists of one server with direct access to a resource (file system, printers, CPU time). The server then allows several clients to access the resource. A file system is a typical resource which can be shared over a network, and Sun's NFS is the most widespread protocol for file system sharing. An important feature of NFS is its complete transparency to the application using it. The application has no need to know whether it is accessing a file on a local file system or from a file system shared over a network.

NFS, designed by Sun several years ago, does not address the security issues arising in this context. NFS is simple in structure and assumes a strong trust model: that is, the user trusts the remote file system server and the network with his data. This poses several risks. The data on the server are available to the server superuser; also, users on the network may assume other identities by changing their IP numbers or their user IDs, allowing data to be read while it travels on the network. Because of this, it is necessary to address the security issues by protecting the data while stored on a remote server and during network transfers.

TCFS (Transparent Cryptographic File System) has been developed at the Dipartimento di Informatica ed Applicazione of the Universita' di Salerno (Italy) and is currently available for Linux. You can look at TCFS as an extended NFS. It acts just like NFS, but allows a user to protect his/her files using encryption.

TCFS requires an NFS server running Linux with the EXT2 file system. It must be used with 2.0.x kernels, since it is based on Olaf Kirch's NFS module. TCFS can be used as a kernel module (and inserted using the **insmod** utility) or can be

compiled into the kernel. When you start the TCFS module or when you boot (if TCFS is statically linked), you will find four copies of the **tcfsiod** daemon running.

TCFS works as a layer under the VFS (Virtual File system Switch) layer, making it completely transparent to the applications. The security is guaranteed by means of the DES (data encryption standard) algorithm. Keys are kept in a special database (/etc/tcfspasswd) which stores keys encrypted with the user's login password. To maximize the level of security, it is best to keep to a minimum number of trusted entities. A TCFS user needs to trust only the kernel and the superuser of the client machine accessing the data. We stress that this minimal level of trust is necessary, since you cannot protect your data from the kernel and the superuser. Both can access memory any time that they want. Our trust model fits perfectly the typical scenario in which TCFS is used: a network of workstations with limited disk space, each used almost exclusively by a limited number of users (you can even think of each user as the superuser of his/her own workstation) and a remote file server sharing files with all the workstations.

In designing TCFS we were interested in providing a robust security mechanism at the lowest possible cost to the user. The security mechanism must guarantee that secure files are not readable:

- by any user other than the legitimate owner,
- by tapping the communication lines between the user and the remote file system server,
- by the superuser of the file system server.

We also protect sensitive meta data—for each file; not only the content but also the filename is encrypted. We hide internal file data dependencies using a DES in the chaining block cipher.

In TCFS, security acts in a transparent way. Secure files can be accessed in the same way as local files—the user has only to authenticate himself to TCFS before starting to work. A special flag, which looks like an EXT2 extended attribute, marks encrypted files to make them distinguishable from unencrypted ones. Thus, TCFS is able to store both secure and unsecure files on the same file system depending on whether or not this flag is set.

We give special attention to making TCFS completely transparent to the file server. Transparency allows the superuser on a server to perform all administration duties in that we don't change the data structures of the file system itself. Special work is needed for a directory with the secure flag enabled. Files in a secure directory are stored with encrypted filenames, and

new files inherit the secure flag, so that they too are secure. Since TCFS acts like a file system in a VFS (virtual file system) layer, standard system calls can be used to access files on the TCFS. No special flags are needed by the **open()** or **create()** system calls. For this reason, all applications can use the new features without being recompiled.

The Working of TCFS

To explain the mechanics of TCFS we will first review the working of NFS. NFS is a simple distributed file system that allows a file system server to *export* its file systems to several clients. Applications, running on a client, access the remote file system via the usual system calls. The client kernel checks to see whether the requested data is on a local file system or an NFS file system. In the latter case, the kernel issues a request to the server; for example, if the application needs to read a block from a file on a remote file system, the client operating system issues a read request to the server. The server, upon receiving a read request, reads the data from its local file system and sends it to the client, which then passes the data to the application. It is important to remember that NFS provides a minimal form of user authentication. The server receives from the client the uid of the user requesting the data and checks if that user is allowed to access the file containing the data. Thus, it is possible for a user to change his **uid** on the client (for example, the superuser of the client machine can use the command **su** to become any user) or to modify the NFS client so that a different **uid** is provided with the request.

When using TCFS, files can be stored in encrypted form on the server file system with a different encryption key for each user. The encryption key is provided by the user to the TCFS client through the **tcfslogin** utility. (A detailed description of the TCFS utilities appears below.) Reading a block of data from the server is achieved following the NFS protocol, with one important exception: once the requested block is received by the TCFS client, it is decrypted before being passed to the application. Similarly, a block of data written by an application is encrypted with the user's key before being passed to the TCFS server. During a read or write operation the user's encryption key never leaves the TCFS client, and data travels between server and client only in encrypted form. Moreover, this approach addresses the problems related to user authentication. While it is still possible for a user to impersonate the legitimate owner of a file, he will receive only encrypted data.

Installing TCFS—Server Side

Set up your TCFS server just as you would an NFS server—by exporting the file system you wish to share with your clients. Usually this is done by editing the /etc/exports file and restarting the NFS daemons (rpc.mountd, rpc.nfsd).

Retrieve **xattrd** from the TCFS package. It can be found in the linux/fs/tcfs/contrib/xattrd directory of the TCFS distribution. Copy **xattrd** to the daemon directory, usually /usr/sbin, and add it to your rc files. For the Slackware distribution, edit the /etc/rc.d/rc.inet2 file to look like Listing 1. rc.inet2 File.

For Red Hat or any other distribution using the System V init script model, create a file in the rc directory (/etc/rc.d/init.d in Red Hat 4) for starting and stopping the **xattrd** daemon and make symbolic links in the rc\#.d directories to start it. In Red Hat you can do this using the **tksysv** script. For an example of building the **xattrd** rc scripts, see Listing 2. File for Building xattrd Script.

Now, reboot the system or run **xattrd** as **root** to prepare the server for TCFS. Notice that **xattrd** reads /etc/exports at startup and so if you change /etc/exports, you must restart **xattrd**. **xattrd** adds functionality to the NFS server and is not meant to be a replacement; therefore, it is possible to use the same server as both a TCFS server and an NFS server.

Installing TCFS—Client side

Installing TCFS on the client is somewhat more complex, since most of the work is performed by the client—the kernel must be rebuilt to support TCFS.

The TCFS distribution provides a tar file to be unpacked in the /usr/src directory. We assume that the kernel sources are in the /usr/src/linux directory (this is the standard for most Linux distributions). Install TCFS with the following steps:

1. Untar TCFS to create the directory /usr/src/linux/fs/tcfs which contains the code for TCFS and its related utilities.
2. Apply the tcfs.diff patch found in /usr/src/linux/fs/tcfs/patches to the kernel. Do this by changing directories to /usr/src and then typing **patch < linux/fs/tcfs/patches/tcfs.diff**.
3. Recompile the kernel. In the FileSystem section you will be asked about TCFS. It is possible to install **tcfs** as a module or built-in. In both cases it is necessary to recompile the kernel following the usual procedure.
4. Install the utilities. Once a kernel with TCFS support has been installed, change directory to /usr/src/linux/fs/tcfs/contrib/binaries where you will find the binaries for the TCFS utilities, and type **make<\!s>install**. It is also possible to compile the source code for the TCFS utilities located in /usr/src/linux/fs/tcfs/contrib/src.
5. Enable use of TCFS. The superuser of the client must generate a key for each user using the **tcfsgenkey** utility. This requires the user's password, so it must be done with the help of the user. The utility **tcfsgenkey** builds a database (/etc/tcfspasswd) where the keys used to encrypt files are

stored. These are kept encrypted using the user's login password as key. In future releases of TCFS we are planning to provide support for smart cards, thus dispensing with the need of keeping the key (albeit in encrypted form) on the client.

6. TCFS utilities. The command mount provided by TCFS is capable of handling TCFS mount operations. We encourage you to use our version of **mount** in place of the standard **mount** command, since TCFS needs some information that the standard **mount** doesn't provide. To mount a file system with TCFS, type:

```
mount -t tcfs server:/remotepath /localpath
```

TCFS also supplies the **passwd** command which is used to update the encryption key database. It acts like the standard **passwd** command, but also updates the tcfspasswd file after changing the password. Changing your password with the old **passwd** command will result in the wrong encryption key being extracted from /etc/tcfspasswd, and thus, in a complete loss of data.

After login, each user has to execute **tcfslogin**. Utility **tcfslogin** requests the user's login password to decrypt the encryption key and pushes the cipher key in the kernel module. To remove the key, the user must execute the utility **tcfslogout**, which needs a TCFS file system mounted to work. In future releases we plan to include support for PAM (Pluggable Authentication Module), making it unnecessary to input the login password twice.

The **lsattr** and **chattr** commands act just as they do in EXT2. The TCFS versions support a new flag, **X**, to enable encryption of files. You can change status of a file (encrypt or decrypt) by typing:

```
chattr $ <+|-> $X
```

An Example

Suppose you have a server named **foo** and a client named **bar** and suppose you export the directory tree named /exports from server **foo** to client **bar**. For this to be true **foo** must have the following line in /etc/exports:

```
/exports        bar(rw,insecure)
```

Now, login as **root** on **bar** and **mount** /exports by typing:

```
mount -t tcfs foo:/exports /mnt/tcfs
```

This command causes the remote file system, /exports@foo, to be mounted on the local file system, /mnt/tcfs@localhost, via a TCFS layer.

Now, suppose your login is **usdm1**, and you own a directory named /mnt/tcfs/usdm1. Login as **usdm1** on **bar** and execute **tcfslogin**; doing so enables you to use encryption in your directory /mnt/tcfs/usdml. If **tcfslogin** is not issued, a permission denied error will be issued when attempting to access files with the **X** flag set.

Performance

In order to evaluate the overhead introduced by encryption of the data sent over the network, we performed a set of tests. We ran the test in the following framework:

- The client machine running TCFS on the Linux 2.0.23 kernel is a Cyrix x686 166MHz processor
- The server machine running as the NFS+xattrd file server is an Intel Pentium 133MHz processor with a 2GB fast SCSI disk.

Since encryption/decryption is a CPU-bound task, having a fast client to perform encryption results in better performance. TCFS makes use of standard VFS caches—no special caching is needed.

The real time values shown in Table 1 and Table 2 were obtained using the **time** command with the following options set for the read operations:

```
time dd bs=xxx if=file of=/dev/null count=n
```

and for the write operations:

```
time dd bs=xxx if=/dev/zero of=file count=n
```

The tables show the following results:

- The overall performance of TCFS for write operations is close to NFS performances plus DES overhead. In the write, we suffer due to the lack of a cache system, since data are written directly to the server file system.
- The performance of TCFS for read operations seems to hide part of the DES time, since VFS caches reduce server I/O.
- Some extra cost is paid by TCFS for I/O of unencrypted files due to handling of extended attributes. In NFS several **getattr** calls are needed to update inode caching. In TCFS we need a **getattr** and a **geteattr** to update inode caching. This causes some extra overhead in TCFS I/O.

Use of other ciphers will result in different performances. We are planning to use IDEA, RC5 and other ciphers as optional modules for TCFS.

Resources

Ermelindo Mauriello (ermmaui@globenet.it) was born in Avellino, Italy on December 10, 1972. He is a computer science student at the Dipartimento di Informatica ed Applicazioni “Renato M. Capocelli” of the Universita' di Salerno in Italy. He has been working on the TCFS project since 1995.

Archive Index Issue Table of Contents

Advanced search

Wrap a Security Blanket Around Your Computer

Lee E. Brotzman

Issue #40, August 1997

TCP_wrappers provide a simple, elegant and effective means to safeguard your network services.

With the Internet growing by leaps and bounds, the security of computer systems has become a major focus of large corporations, small businesses and individuals alike. Hardly a week goes by without a security flaw being discovered in some network. Many companies are reducing the threat by installing firewalls between their internal networks and the Internet, but this option is generally too expensive and cumbersome for single users running Linux from home or office. TCP_wrappers was written by Wietse Venema, Eindhoven University of Technology, The Netherlands. It provides a simple, elegant and effective means to safeguard your network services from being accessed and possibly abused by intruders.

In this article we will discuss what TCP_wrappers are and how they work, and how to configure TCP_wrappers to protect your machine from unauthorized access. We'll also discuss some of the more advanced features of TCP_wrappers which provide detailed logging and can even help track attempts to break into your machine.

What are TCP_wrappers and How Do They Work?

First, we need to know how a transmission control protocol (TCP) connection, such as **telnet**, is accomplished. TCP network connections are based on the "client/server" model. The **telnet** program is a client that communicates with a server program, or daemon, called **telnetd** or **in.telnetd**, depending on how your machine is set up. Since most Linux distributions use the name **in.*serviced*** in the directory /usr/sbin for network daemons, I will use that naming convention for the remainder of this article.

All requests for network services first go through the “Internet daemon”, **inetd**. (As with all things in life, there are exceptions to this rule, see “What TCP_wrappers Cannot Do” below.) This daemon uses two configuration files to determine how to respond to requests for network connections. The file /etc/services lists the names of particular services and the port numbers for those services. The file /etc/inetd.conf lists the names of the services and the names of the programs or daemons providing the services. Listing 1 and Listing 2 contain some sample lines from the /etc/services and /etc/inetd.conf files.

If I sit down at **my.linux-box.com** and type the command:

```
telnet your.machine.com
```

My **telnet** client sends a packet of information containing (among other things) the Internet address of the source **my.linux-box.com**, the Internet address of the destination **your.machine.com** and the port number for the connection. The port number for **telnet** is 23. **inetd** looks up port 23 in /etc/services and finds the service name **telnet**. It then looks up **telnet** in /etc/inetd.conf and finds that it needs to run the daemon called **in.telnetd**, listed in the rightmost column of Listing 2. **inetd** runs **in.telnetd** connecting it to port 23 and then goes about the business of listening for more connections. **in.telnetd** responds to my client, asking for a user name and password and starting up the telnet session.

What if you don't want anyone else to **telnet** into your computer? You can modify the code for **in.telnetd** to look at the source address of the connection request and to reject any addresses from outside your local machine or domain. If **telnet** were the only network service this would be easy, but there are dozens of network services, and it would be a nightmare to modify every daemon to limit access to your machine.

At this point TCP_wrappers comes to the rescue. The wrapper program is a tiny daemon that stands between **inetd** and network daemons such as **in.telnetd** and **in.ftpd**. Since all TCP connections are started up in basically the same fashion, a single wrapper program can be used to control access to almost all TCP network services.

When wrappers are installed, the Internet daemon is reconfigured to run the wrapper instead of the ordinary network daemon. The wrapper checks the source address of the connection and the service it wishes to connect to and decides whether to allow the connection. If **your.machine.com** denies my request for a **telnet** session, all I see is a dropped connection. If the request is allowed, everything proceeds normally, and the wrapper never actually interacts with my **telnet** client. In either case, the wrappers write a note in the system logs to let you know whether I was successful in connecting to your machine.

Installing TCP_wrappers

Every major Linux distribution comes with TCP_wrappers installed as part of the networking package. To see if you have TCP_wrappers installed on your machine, look in the /etc directory for two files called hosts.allow and hosts.deny. These are the configuration files used by the TCP_wrapper daemon, **tcpd**. You can also look at your /etc/inetd.conf file for lines like this:

```
telnet stream tcp nowait root /usr/sbin/tcpd \
        in.telnetd
```

The **telnet** option **/usr/sbin/tcpd** indicates that whenever someone tries to **telnet** to your machine, he will first connect to the TCP wrapper.

TCP_wrappers is probably on your Linux system, so I won't go into the process of compiling them from scratch in this article. See the sidebar “Where to Get TCP_wrappers” for more information on how to download and install TCP_wrappers.

Configuring TCP_wrappers

As I mentioned above, the TCP_wrappers daemon **tcpd** gets its instructions on whether to allow or deny access from the two files /etc/hosts.allow and /etc/hosts.deny. **tcpd** first scans /etc/hosts.allow for “rules” that match the particular service and computer host name, then searches /etc/hosts.deny. If no match is found, access is allowed. By default, most distributions ship TCP_wrappers “Completely Open”, i.e., the files /etc/hosts.allow and /etc/hosts.deny are empty, allowing access to all Internet services on your machine to everyone.

Before you configure TCP_wrappers, you must decide whether you want your machine to be “Mostly Open” or “Mostly Closed”. Mostly Open means that most services are available to most other computers on the Internet; this is useful for blocking access to just a few troublesome sites or to close off one or two services. Mostly Closed means that most services are closed off to most other computers. For personal computers with normal at-home usage, Mostly Closed is probably the way to go, and it is certainly the more secure option.

Let's look at a “Mostly Open” configuration first. Since we are allowing most connections, the /etc/hosts.allow file is left empty. In the **/etc/hosts.deny** file, let's put in a rule to deny access to **telnet** and **rlogin** to anyone coming from the machine **nasty.badguy.net** and anyone in the domain **cracker.org**. The requisite line in /etc/hosts.deny would be:

```
in.telnetd in.rlogind : nasty.badguy.net \
        .cracker.org
```

Note the leading "." in front of **cracker.org**. It signals **tcpd** to deny access to any machine in that Internet domain. Since the crackers at these sites probably know how to exploit network services other than **telnet** and **rlogin**, you can deny access to all services using the wild card **ALL**:

```
ALL : nasty.bad-guy.com .cracker.org
```

Other wild cards that can replace specific host names include **LOCAL, UNKNOWN, KNOWN** and **PARANOID**. **LOCAL** matches any name without a "." in it, i.e., host names in your local domain. **KNOWN** and **UNKNOWN** refer to hosts either found or not found in the Domain Name Service, respectively. **PARANOID** matches any host whose name does not match its Internet address. This option is not often used, since the wrappers are compiled to reject access to any host that matches this condition before checking the hosts.allow and hosts.deny files. To allow access to all network services to machines in our local domain, we can put the following line in /etc/hosts.allow:

```
ALL : LOCAL
```

Now let's look at a "Mostly Closed" configuration. Remember that hosts.allow is checked first, then hosts.deny and, finally, access is allowed only if no match is found in hosts.deny. To close all services to all outside machines, we use the following rule in our hosts.deny file:

```
ALL : ALL
```

In hosts.allow we list only those specific services we want others to use. Of course, we still want to access all of our own services on our own machine. Suppose that we also want to **telnet** into our machine from a shell account provided by our Internet Service Provider at **my.isp.net**, and we want to allow anyone to **finger** our accounts. The rules to put in the /etc/hosts.allow file are:

```
ALL        : localhost
in.telnetd : my.isp.net
in.fingerd : ALL
```

Now, if we would also like to keep the crackers from **cracker.org** from using **finger** to get information about us, we can modify this:

```
ALL        : localhost
in.telnetd : my.isp.net
in.fingerd : ALL EXCEPT .cracker.org
```

As you can see, there is quite a bit of flexibility—but with this flexibility comes the possibility of confusion and, even worse, error. If the configuration files for TCP_wrappers are wrong, you may *think* you are secure, when in fact you are not. To check your configuration and test its protection, Wietse Venema provided two additional programs: **tcpdchk** and **tcpdmatch**.

tcpdchk checks the configuration files for any problems. It can tell if you have used wild cards like **ALL** or **LOCAL** incorrectly, if there are nonexistent host

names in the access rules, if there are rules for services controlled by **tcpd** in the /etc/inetd.conf file and much more. For example, the output from **tcpdchk** for the above Mostly Closed configuration on my machine yielded the following information:

```
# tcpdchk -v
Using network configuration file: /etc/inetd.conf
>>> Rule /etc/hosts.allow line 6:
daemons:  ALL
clients:  localhost
access:   granted
>>> Rule /etc/hosts.allow line 7:
daemons:  in.telnetd
clients:  my.isp.net
warning: /etc/hosts.allow, line 7: my.isp.net: \
        host not found
access:   granted
>>> Rule /etc/hosts.allow line 8:
daemons:  in.fingerd
clients:  ALL EXCEPT .cracker.org
access:   granted
>>> Rule /etc/hosts.deny line 10:
daemons:  ALL
clients:  ALL
access:   denied
```

I used the **-v** switch for **tcpdchk** to generate more verbose output. Note that the program says **my.isp.net** was not found, which is perfectly true, since it is a host name made up for this example. Also, note that I did not get a similar message for the equally fictitious **.cracker.org**. That is because it is for an entire domain, and **tcpdchk** doesn't check if a domain is registered, but rather if a particular host name is in the DNS.

tcpdmatch tests your configuration against a *virtual* request for an Internet connection. You provide the name of the daemon and a host name, and it tells you whether that connection would be allowed or denied. For example, if I would like to know if the webmaster at www.linuxjournal.com can finger users on my system, I would enter the following:

```
# tcpdmatch in.fingerd webmaster@www.linuxjournal.com
client:   hostname www.ssc.com
client:   address  199.184.169.67
client:   username webmaster
server:   process  in.fingerd
matched:  /etc/hosts.allow line 8
access:   granted
```

Note that **tcpdmatch** found the real host name of www.linuxjournal.com to be www.ssc.com and reports its Internet address. The last line tells me that **finger** is indeed allowed from this host.

In *Practical UNIX and Internet Security, Second Edition* by S. Garfinkel & G. Spafford, O'Reilly & Associates, 1996, the authors state:

> Programs like *tcpdchk* and *tcpdmatch* are excellent complements to the security program *tcpwrapper*,

> because they help you head off security problems before they happen. Wietse Venema is to be complimented for thinking to write and include them in his *tcpwrapper* release; other programmers should follow his example.

I wholeheartedly agree.

Tracking Usage with Wrappers

TCP_wrappers write a message into your system logs every time a connection is requested, whether it is granted access or not. These entries in the logs above are reason enough to have TCP_wrappers installed on your system. The messages are written through the standard syslog facility and by default go to the same place as mail transactions. On my Linux distribution, Caldera Network Desktop based on Red Hat Linux, the default has been changed so that the messages are written to the same log file as other daemons (LOG_DAEMON facility).

In any event, when someone accesses my machine via **telnet**, a message like this is placed in the /var/log/messages:

```
Apr  9 17:24:58 ads in.telnetd[15339]: connect from somewhere.else.com
```

If the connection was refused, the message would read:

```
Apr  9 17:25:15 ads in.telnetd[15604]: refused connect from someother.place.com
```

If I want to see all the **telnet** attempts in my log, I can simply type the command:

```
grep telnetd /var/log/messages
```

TCP_wrappers can give me even more information through the use of “booby traps”. TCP_wrappers can be configured to *run shell commands* when certain services are requested. Let's assume I have reason to suspect someone at **nasty.badguy.com** is trying to use the Trivial FTP program (TFTP) to steal my password file. In my /etc/hosts.deny file, I can put the following line (this example is straight from the **hosts_access(5)** man page that comes with TCP_wrappers):

```
in.tftpd : nasty.bad-guy.com : ( /usr/sbin/safe_finger -l @%h |\
        /bin/mail -s %d->%h root) &
```

Access to TFTP is denied to all users from **nasty.badguy.com**. In addition, the command:

```
safe_finger @nasty.badguy.com
```

is run, and the results are piped into a mail message sent to the root user with the subject line:

```
in.tftpd->nasty.bad-guy.com
```

safe_finger is a command provided along with the TCP_wrappers that strips out any “bad” characters, like control sequences and data overruns. Running **safe_finger @hostname** generates a list of everyone currently logged into that system. The strings **%h** and **%d** are called expansions, and **tcpd** replaces them with the corresponding text for the host name and daemon process, respectively. Other expansions include **%a** for the client Internet address and **%u** for the client user name.

Now, this isn't a perfect solution, since our cracker friend may have disabled his finger service or altered it to give false information; however, this example does show us the power of the TCP_wrappers program.

Using Advanced Options

The access control language in the /etc/hosts.allow and /etc/hosts.deny files is quite simple, yet powerful, but TCP_wrappers can be compiled to include even more powerful extensions to the normal access controls. Instead of having the line:

```
service : host
```

in the configuration files, you can have the line:

```
service : host : option : option ...
```

Where **option** can be **allow**, **deny** or many other advanced options.

To enable the advanced options, compile the wrapper programs with the **-DPROCESS_OPTIONS**. The wrappers are compiled in this way by my Caldera/ Red Hat distribution. To check your distribution for the advanced options, run **tcpdchk<\!s>-a**. On my system, I see the following output:

```
warning: /etc/hosts.allow, line 6: implicit "allow" at end of rule
warning: /etc/hosts.allow, line 7: my.isp.net: host not found
warning: /etc/hosts.allow, line 7: implicit "allow" at end of rule
warning: /etc/hosts.allow, line 8: implicit "allow" at end of rule
```

The message **implicit<\!s>"allow"** indicates my version of the wrappers is looking for additional options in the /etc/hosts.allow file. If your distribution does not have PROCESS_OPTIONS compiled in, you will not see this message.

Using the advanced options, we can do away with the /etc/hosts.deny file entirely, since the options for **"allow"** and **"deny"** can be added to any rule. We can change the logging level, control the priority (“niceness”) of the network service, look up user names with the RFC 931 “ident” protocol and display customized “banners” for each service.

More information on these advanced features can be found in the **hosts_options(5)** man page included with TCP_wrappers, or in the Garfinkel & Spafford book cited above (a must-read for anyone concerned with network security). For now, let's convert our /etc/hosts.allow and /etc/hosts.deny files to use the advanced options. The etc/hosts.deny file is no longer needed. In /etc/hosts.allow we rewrite the rules as follows:

```
ALL        : localhost               : allow
in.telnetd : my.isp.net              : allow
in.fingerd : ALL EXCEPT .cracker.org : allow
in.tftpd   : nasty.bad-guy.com       : spawn \
        (/usr/sbin/safe_finger -l @%h |\
        /bin/mail -s %d-%h root) &
                                     : deny
ALL        : ALL                     : deny
```

In English, this says the following:

1. All services from our own machine are allowed.
2. **telnet** is allowed from **my.isp.net**.
3. **finger** is allowed from everywhere except hosts in the **cracker.org** domain.
4. **tftp** is not allowed from **nasty.badguy.com**, and if they try, we will spring a "booby trap" to find the guilty party.
5. All other services from everywhere else are denied.

What TCP_wrappers Cannot Do

As we have seen, TCP_wrappers provide a simple and effective means to control access to our machines. However, we must still remember "There is no secure in computer security, only more secure or less secure." As with all security measures, TCP_wrappers have their limitations.

First and foremost, wrappers cannot control access for those services started at boot-time and run until system shutdown. Services like **sendmail** and **httpd** (the World Wide Web server) fit this category. These services are always listening to their own ports and require their own access controls. Discussions of the security of **sendmail** and the World Wide Web fill entire volumes and are certainly outside the scope of this article.

TCP_wrappers may also be vulnerable to "host name spoofing". Services like **rsh** and **rlogin** depend on the host name being correct. If you use a DNS server on which you cannot look up host names, it is possible for an attacker to "spoof" the name lookup by hiding his computer's name behind one your machine "trusts". You can thwart these attacks by putting an entry for the Internet address and host name in your local /etc/hosts file, so that you do not depend on outside DNS lookups (an added benefit is that host name lookups are a lot faster). Be aware that you are now responsible for keeping the /etc/hosts file up

to date. If a computer in the /etc/hosts file changes its Internet address, access will be denied until you change its entry. Fortunately, this is a rare event, and I regularly put entries in my /etc/hosts file for computers I contact often and for every host allowed access to my machine.

TCP_wrappers also do some additional homework to avoid name spoofing attacks. When compiled with the default option **PARANOID** (see the discussion of wild cards above), the wrappers not only check an Internet address by looking up its name but also by looking up its address. If the two don't match, access is automatically denied.

Another vulnerability can come from “source routing”, a situation where a computer from some “outside” address claims to be a trusted computer on the “inside”. TCP_ wrappers can be compiled with **KILL_IP_OPTIONS** to disable source routing. Luckily, we Linux users generally do not have to worry about this sort of attack, since IP source routing is turned off by default in the kernel itself.

Finally, even though you can use wrappers to control access to certain services, the best way to avoid exploitation of a service is to completely shut it off from the beginning. If you have no use for **rsh** or **rlogin**, edit your /etc/inetd.conf file and put a hash mark, **#**, in front of the lines for the **shell** and **login** services. This advice goes for any other service you don't use. Security holes cannot be exploited on services that are never started. “When in doubt, comment it out” is my motto.

Conclusion

TCP_wrappers are cheap and effective tools for controlling access to your Linux computer. Even without employing the access control features of wrappers, the ability to trace each and every connection to your machine through your system logs can be extremely valuable. TCP_wrappers can control access with a broad brush or a single pen stroke. Either way, I hope this article has raised your awareness of the ease with which you can control the “network face” of your machine.

Where to Get TCP_Wrappers

Lee Brotzman is the Vice President of Advanced Data Solutions, a consulting firm in State College, Pennsylvania. He currently works as an instructor in Internet security, and has presented courses in Unix system security at many U.S. Government facilities. He also serves as a consultant in the design and development of networked information systems and electronic publishing. He resides in State College with his wife/business partner of fifteen years, their three children, one dog, two cats and a goldfish that thrives on dog biscuits

(which makes the cats extremely nervous). He can be reached via e-mail at leb@vicon.net.

Archive Index Issue Table of Contents

Advanced search

Programming with the XForms Library, Part 2: Writing an Application

Thor Sigvaldason

Issue #40, August 1997

We learn to write an application with XForms by simulating a game with 2 players and 2 actions.

Last month we began this series on XForms by explaining how to install the forms library and include file. We also took a stab at programming with XForms by writing a couple of simple programs. In this month's article, we're going to write a fully fledged application. We'll start with an explanation of the project, and then see how to implement it with XForms.

The Project: A Game Theory Simulator

Our task is to implement a game theory simulator. If you don't happen to have a doctorate in mathematics, you might want to have a look at *A Primer on Game Theory* which appears on page 52 of this magazine. We're attacking a non-trivial programming task in order to get a handle on how to do “real-world” programming with XForms. It's not important that you understand every nuance of game theory, since our main goal is to figure out XForms.

In a formal game, there are two main entities we have to consider: players and payoffs. So our simulator should allow us to set relevant values for these elements. Players, for example, are defined by actions and strategies. Similarly, payoffs are just a set of values that players receive when they play the game. As a great simplifier, we'll assume there are only two types of players and only two possible actions. This reduces the dimensionality of our programming problem. A good exercise for readers would be to try and relax the two strategy limitation, but make sure you understand the initial program fairly well before trying to modify it.

Since we're creating a graphical application, we'll want a point-and-click interface for setting up our players. The method adopted is to think of players as having a finite number of states. In every state, there is an action to be taken. Since we've limited ourselves to just two possible actions, let's call these A and B. Our simulator will be used for repeated games, so we want to be able to design players who can change their actions. That is, move to another state which tells them to play a different action. There are only two types of players, Column Players and Row Players, and two actions, so the choice of which action to play can be affected only by what the other player did.

Let's say you want to design a player who always plays action A. That's simple; just set the action in each state to A. A more complicated example would be to design a player who plays A if the other player chose A last period, but B if the other player chose B. This is easier to see with a diagram:

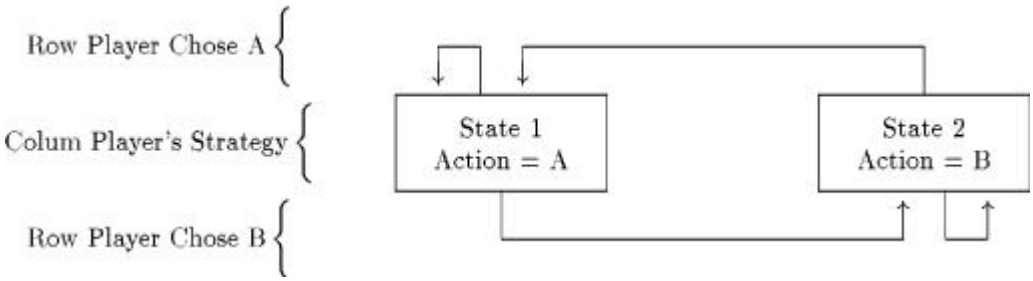


Figure 1. Diagram of Player States

Here we see that the transitions from state to state can be made contingent on the behavior of other players. So to implement an interface for designing players, we have to be able to specify the action in each state and the transitions to perform, i.e., which state to jump to, given the behavior of the other player. We cannot make one player's choices contingent on what the other player does in the current period. This would violate one of the tenets of standard game theory: simultaneous choice of action. Here we show only two states, but we'll allow for more complicated player strategies in the actual program.

We'll also want several players of each type to exist and to be randomly matched against players of the other type. This sounds more difficult than it is, since we still have to design only two types of players. The population of Row Players, for example, should differ only in what state they are currently in, not in the overall design of their strategy.

Like player design, we'll want an easy way for the user to set and edit payoffs. This is a little simpler, since we just need a graphical representation of the payoff table and a method for letting the user change these values. We'll want both of these features to appear in their own windows so we can pop them up when we need them.

Once the user has specified player strategies and payoffs, we'll also need a way to actually run the simulation. This routine should match the players, allocate their payoffs, and handle the transitions from state to state. It should also let us set how long the game should run, and give us some nice visual feedback on the progress of play.

All of this input, interaction and editing may seem very complicated. It would probably require a fairly cumbersome set of menus or a command language, if we were going to program this project on a simple terminal window. But with XForms, we can easily create windows, input fields and other graphical elements required to implement our game theory simulator. If this is all a little hazy, it may be a good idea to play with the running program a little (see below), and then come back to this section.

The xgtsim Program

Let's just plunge right in. We'll get an example up and running right away, and then use the rest of this article to explain how it works. The program is called xgtsim and the C source code can be found in Listing 1.1 Although you're welcome to type it in, it's also available on the web site for this series (see http://a42.com/~thor/xforms, Listing 1). It should compile with the command:

```
gcc -lX11 -lforms -lm xgtsim.c -o xgtsim
```

From within the X Window System, you should be able to run the program by typing **./xgtsim** in an xterm window. If you have problems, you may want to go back and review last month's article on installing XForms. With all possible windows open, the running program should look something like Figure 2.

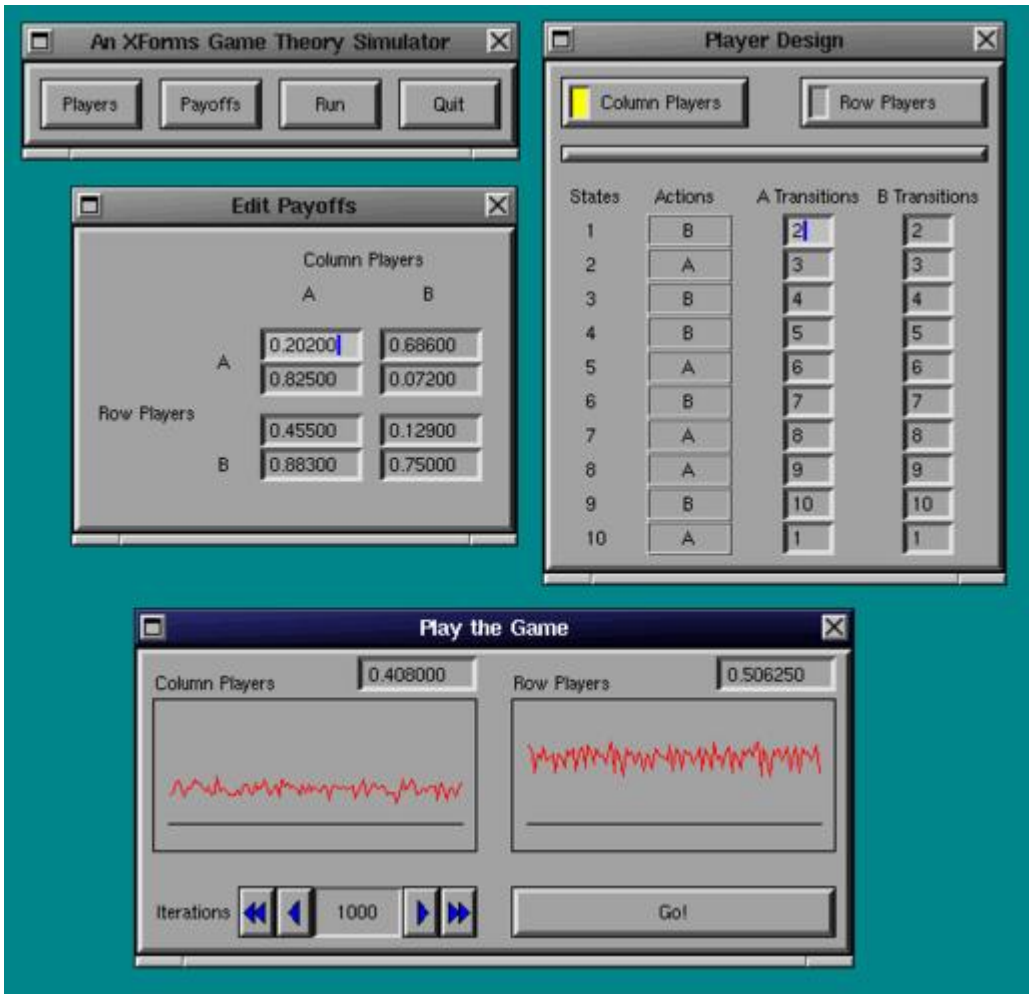


Figure 2. xgtsim with Windows Open

If you want to play around with the program before continuing with the rest of this article, one useful exercise would be to set up a prisoner's dilemma. Just use the payoff editor to set values the same as they appear in the primer, and then try some different player strategies. In particular, try and figure out what happens when two Tit-for-Tat strategies come up against each other. Does it matter what initial strategies they play?

The Flow of the Program

Last month, we saw that the basic steps to designing an XForms program are as follows:

1. Include forms.h to access the XForms routines
2. Call **fl_initialize()** as soon as possible
3. Set up your graphical interface by creating forms
4. Assign actions to relevant objects by setting callbacks
5. Show one or more forms
6. Turn control over to **fl_do_forms()**

We use this approach in xgtsim. Like all C programs, execution begins in the **main()** routine, which is at the very end of the source code. First we call **fl_initialize()** to set up XForms, and allow it to parse command line arguments. Next, we call **set_defaults()** which seeds the random number generator, and sets some default values for our payoffs and player design variables: **payoffs[][][]**, **state_actions[][]** and **state_transitions[][][]**.

A call to **create_forms()** is then made, which sets up all of our windows, graphic elements and callbacks. We'll go into more detail shortly, but let's look at how this works for the simplest case: quitting the program. Within the **create_forms()** code, we use **main_window** (a variable of type **FL_FORM**) to create a window which will be shown when the program starts up. This window has four buttons on it, called Players, Payoffs, Run and Quit. Notice that the Quit button is set to call the function **quit_xgtsim()** with the command:

```
fl_set_object_callback(obj, quit_xgtsim, 1);
```

This means whenever the mouse is clicked on the button labeled Quit, the quit routine will be called. This function, in turn, simply calls **fl_finish()** and then exits.

To return to the flow of the **main()** function, after setting up all of our windows, buttons and so on with **create_forms()**, we then make our **main_window** appear with a call to **fl_show_form()**. Then we turn control over to the user by calling **fl_do_forms()**.

It's crucial to understand that setting up our forms in **create_forms()** does more than just decide how graphics should be laid out on the screen. By setting callbacks to link button pushes and data inputs with specific actions, we've actually set up the whole flow of the program. When the user pushes the Payoffs button, it is XForms (via **fl_do_forms()**) which calls the relevant routine to make the Payoffs window appear, and to handle subsequent interaction with that window. In fact, if we've set all our callbacks correctly, execution never returns to **main()**. The **fl_do_forms()** routine returns only if the user activates an object which does not have a callback associated with it.

Some Details

Since **create_forms()** is so important, lets look at it in more detail. We use and re-use a generic pointer called ***obj**, which is of type **FL_OBJECT**, to create many of our graphical elements. This can be a little confusing, but we'll clarify things as we go.

The first form created in **create_forms()** is **main_window**. This is a global pointer variable which we declare early in the source code. We tell XForms it is a

window which should be 290 pixels wide and 50 pixels high with the assignment:

```
main_window = fl_bgn_form(FL_NO_BOX, 290, 50);
```

In the following nine lines of code, we create four buttons which will be used to pop up windows for user interaction and to quit the program. Each time we need a new graphical element, we just use **obj**, which saves memory and keeps things simple. Just remember that whenever we reassign **obj**, all subsequent functions passing **obj** as a value will affect the most recent assignment. The Players, Payoffs and Run buttons are all linked by a callback to a routine called **display_forms()**, but they are set to call that routine with the values 1, 2 and 3 respectively. The **display_forms()** routine, in turn, uses these values to decide which window to display. After creating the Quit button, we tell XForms we're done adding elements to this form by calling **fl_end_form()**.

We then go on to create the **player_window**, **payoff_window** and **run_window**. These all follow the same general pattern; declare the dimensions of the window with **fl_bgn_form()**, add as many objects as we want (assigning callbacks as we go), and then finish with **fl_end_form()**. We'll look at the **run_window** in detail, since it's the simplest. Once you have it figured out, you'll probably want to look over the other two on your own.

Since we want visual feedback from the game, we create two charts in the **run_window**. We make these into line charts by specifying **FL_LINE_CHART**, and we set the dimensions by including 4 integer values. The first two values represent where the upper left corner of our chart should appear, with 0,0 being the very top left corner of the form the object is being created on. The next values describe the width and height of the object. Finally, we supply a string to give the chart a label:

```
column_chart = fl_add_chart(FL_LINE_CHART, 10, 30,
        190, 90, "Column Players");
```

You may be wondering why we assign this function call to a variable called **column_chart** instead of using our generic **obj** variable. This is done because **column_chart** is declared as a global variable, which is accessible to all the routines in xgtsim. In particular, when the game is actually being run, the **play_the_game()** routine uses this global variable to add values to the chart we just created—look for the function **fl_add_chart_value()**.

With the label **"Column Players"** assigned to our chart, the default behavior is for it to appear below the chart. We move it to the top left corner with the call to **fl_set_object_lalign()**. Then we limit the number of items which can be displayed with **fl_set_chart_maxnumb()**. We then create an almost identical chart to display information about Row Players.

In addition to chart feedback, we create two browsers to display numerical data. This is accomplished with calls to **fl_add_browser()**. Browsers are very useful objects in XForms, and they can be used in many different ways. Our implementation here is very simple, but you can learn more about them in the XForms documentation.

To allow the user to set the number of iterations the game should run, we create a counter, and set lots of options. First we align the label to appear on the left, then set the precision to 0. This just means we want our counter to hold integer data, since you can't really perform half an iteration. A standard counter appears on the screen with two sets of arrows. Whenever they are pushed, they change the value of the numerical data the counter is holding. We set bounds on this data with a call to **fl_set_counter_bounds()**, and then make one set of buttons change the value by 1 and the other set change the value by 100 by setting the counter step rates. We also set the starting value in the counter to a default value (stored in **numb_iterations**), and then record a callback. Whenever the counter object is changed, the routine **set_iterations()** is called which sets the variable **numb_iterations**.

The run window also contains two buttons, one to start the game running and one to stop it. Notice that we create these two buttons in exactly the same place on the form, so that they are on top of each other. Before finishing, though, we hide the **stop_button** to ensure the **go_button** is visible. When the **go_button** is pushed, it calls **play_the_game()**, which hides the **go_button** and makes the **stop_button** appear. The ability to call **fl_hide_object()** and **fl_show_object()** makes form design in XForms very flexible, since you can design windows where objects appear and disappear according to any number of conditions. When an object is hidden, it is impossible for the user to activate it.

Once **fl_end_form()** is called, we are nearly done with this window. Immediately afterwards though, we call:

```
   fl_set_form_atclose(run_window, close_forms, 0);
```

This tells XForms what routine to run when the window manager sends the close window signal. On most window managers, this signal is sent when the user clicks on a close icon in the title bar of the window in question. This is like a callback, but the format is slightly different. In a normal callback, the declaration is of the form:

```
   fl_set_object_callback(the_object, the_function,
            an_argument);
```

The function pointed to by **the_function** must accept two arguments, an **FL_OBJECT** pointer and a long, as in:

```
the_function(FL_OBJECT *obj, long an_argument);
```

This function must return void. When the window close signal is sent, however, it applies to an entire window/form, not a particular object on that form. So the first argument to **fl_set_form_atclose()** must be a pointer of type **FL_FORM**, as in:

```
close_forms(FL_FORM *form, void *an_argument);
```

This function must return an integer, and in particular, it should return **FL_OK** if you want the window to actually close, and **FL_IGNORE** if you want the window to remain visible.

Having looked at **main()** and **create_forms()**, the rest of the source code is fairly easy to follow. The most complicated part is how the **player_window** uses the **row_or_column** variable to edit both types of players on a single form. The general idea is as follows. The global variables **state_actions[][]** and **state_transitions[][][]** hold all the data on the current state of both types of players, i.e., Column Players and Row Players. On the Player window, there are two buttons allowing the user to choose which type of player they want to edit. Whenever these are pushed, the Player window must be updated to reflect the state of these variables. This is done by the **set_row_or_column()** routine, which reads values from **state_actions[][]** and **state_transitions[][][]** into the relevant objects on the Player window, which are **action_choices[]** and **transition_inputs[][]**.

With the window updated to reflect the current state of the relevant set of players, the user can now edit these values. This is accomplished via the **set_player_values()** function, which is called whenever any of these on-screen objects are changed. We do not bother trying to figure out which object is changed, but simply read all values on the Player window into **state_actions[][]** and **state_transitions[][][]**.

The only remaining subtleties in the program are the use of the **abort_flag** variable and the call to **fl_check_forms()** in the iteration loop of **play_the_game()**. When the Go button is activated in the Run window, **play_the_game()** is called. One of the first things done in that routine is to set **abort_flag** to zero. Players are matched, payoffs made and the charts on the Run window are updated. At the bottom of the iteration loop, we check to see if **abort_flag** has been changed from 0 to 1, and if it has, we stop the run. You may be confused as to how this flag's value could have possibly changed within this algorithm.

The key lies in the call to **fl_check_forms()**. This is a non-blocking routine that works just like **fl_do_forms()**, except that it exits immediately if no objects were activated. This exacts a small performance penalty, since the program is effectively monitoring all its objects while the game is running, but it is well

worth it. Since we set a callback to the Stop button to change **abort_flag** to one (via **stop_the_game()**), clicking on the Stop button will cause the current game to be aborted.

This has the added benefit of allowing us to modify all our data while the simulation is running. For example, we can alter payoff values and immediately see how this changes the unfolding game via the visual feedback in the Run window. Similarly, we can change player strategies on the fly, and watch how this affects their performance. This probing and runtime editing of parameters is often very difficult to achieve with standard C running on a console, but with a few global variables, a little sensible design and a call to **fl_check_forms()**, XForms makes it almost trivial.

Things to Try

If you've managed to pour over xgtsim and get a good feel for what's going on, you may want to try altering the source code to test your understanding. One thing to attempt would be adding an extra button to the Main window that randomizes all the current variables. That is, suppose the user has set payoffs and strategies, but wants to scramble these values. You'll not only have to add the button and set up the callback, but you'll have to update any currently displayed windows to reflect these changes.

The charts in the Run window currently provide only average feedback on the two types of players. Try adding more charts or other elements to display information on the best and worst players in each category.

If you're feeling really ambitious, then try altering xgtsim to allow for more actions and more complicated strategies. This can get very complicated, since elements like the payoff matrix will have to grow and shrink depending on how many actions are currently possible. This can be accomplished by dynamically creating new objects and forms, something we haven't covered so far.

		Column Player: Cooperate	Column Player: Defect
Row Player	Cooperate	(3,3)	(5,0)
	Defect	(0,5)	(1,1)

Payoffs are ordered Column Player, Row Player

Coming Next Month

In playing with xgtsim, you may find a set of strategies and payoffs that generate interesting results. Currently there's no way to save this state of the game, because we have no file-based input and output. We'll be adding that next month, by using XForms pre-built file requester routine. It's just part of a whole set of “goodies” that XForms includes, and we'll be looking at most of them.

Resources

We'll also spiff up our application with some pixmaps, learn how to set gravity parameters to control window resizing, and look at a few other interesting features of XForms.

Thor Sigvaldason is the author of the statistics program xldlas which uses the XForms library (see *LJ* #34, February, 1997). He is trying to finish a PhD in economics, and can be reached at thor@netcom.ca.

Archive Index Issue Table of Contents

Advanced search

Linux Means Business: Security and Authentication with Digital Signatures

Robb Shecter

Issue #40, August 1997

How one university uses PGP and digital signatures to make its network secure.

PGP and public-key cryptography are used all the time for encrypting e-mail and other kinds of messages. They can also be used in other interesting ways. This article describes two other uses for PGP and digital signatures that can help make networks more secure. The University of Maryland University College European Division (quite a mouthful) has 65 Linux-based computer labs in 10 countries. A Linux box in each lab serves files via NFS to Windows/Linux dual-boot clients. The labs are spread out over a huge geographical area, and many are hard to reach. We depend on Linux's reliability to make the system work. At the “education centers”, there is usually no technical support. If a network is down, someone from Computer Field Support must go to the center on an overnight trip to fix it.

To keep things as maintenance-free as possible, we have to develop some secure and reliable systems for managing the networks and the users. The two systems talked about here both use “clear-signed digital signatures” to accomplish their goals. One is a system to securely transmit software upgrades; this has been implemented in Perl and is in use today. The second is a system for remotely authenticating users without the need to access a user database. This one is in the design/specification stage.

There are pointers to information about getting started with cryptography at the end of this article.

Securely Transmitting Software Upgrades

We realized we needed a security system when it came time to upgrade the software on our lab servers. We had to install new versions of the client

programs, make modifications to the server config files and other changes. We knew that, in many cases, the upgrades must be able to be made by people with little computer knowledge. The fact that server system files might have to be modified in a particular upgrade meant that superuser privileges would have to be given out. The three situations we wished to prevent were:

1. Simple media unreliability—the software was going to be delivered via a network connection, on zip drive disks or on conventional floppies. The system would have to protect itself against flaws in the media, such as a floppy disk with bad sectors. The system should refuse to begin the installation if any part of the package is bad.
2. “Man in the middle” attack—in general, an attack in which someone alters the data after it's been sent, but before it's been received. Once the floppies arrive at the education center, they're left lying around on the user's desk for a while. A curious (or devious) student can pick them up and add some special configuration files to be installed. Since superuser access is given to the upgrade program, someone could modify the contents of the packages and gain root access.
3. Unauthorized upgrades—our goal of making the upgrades as easy as possible works for approved *and* unapproved users. An attacker who gets access to one of our upgrade floppies could figure out the file formats and create new upgrades that would change any system files.

These three problems can be summed up as a must to verify integrity and authenticity. We must make sure that the data has not been altered, deleted or added to in any way. We must also make sure that the data comes from the approved source—in this case from our Computer Field Support group. Integrity and authenticity are exactly the functions digital signatures provide. The following protocol solves our problem:

1. Computer Field Support (CFS) generates a public and private key pair.
2. A package file listing is generated.
3. An MD5 checksum is generated for each file and listed in a second column. See Listing 1.

1. This two-column listing is digitally clear-signed with CFS's secret key. This compromises the certificate delivered with the software package. See Listing 2.

1. At installation time, the digital signature is checked using CFS's public key, which is stored on the server.
2. An MD5 checksum is generated for each file in the package and checked against the corresponding string in the certificate.
3. The installation program in the package is executed.

Using this system, a file can't be modified, because the MD5 checksums wouldn't match in step 6. The checksums in the certificate can't be altered, because the certificate's digital signature would fail in step 5. PGP and md5sum are called from shell or Perl scripts to do all the work. The script that creates the certificate is very simple and doesn't require the user to know how to use PGP. All he needs to know is the correct pass phrase to enter:

```
#!/bin/sh
rm listing.asc 2> /dev/null
md5sum * | pgp -staf > listing.asc
```

The user in the field runs another program, which also calls PGP and md5sum. The certificates are more secure when clear-signed than when encrypted, because at this stage we don't rely on any “secrets” being stored on the remote servers. Only the CFS public key is sent into the field. If anyone breaks into one of our computers, the information in the public key is harmless. When we encrypt the certificates, we need to make a second public/private key set for the servers themselves. The private key would be stored on the server and used to decrypt the messages, which would be something “interesting” for crackers to get by. Decrypting the messages also means that a pass phrase must be given to PGP. Either the user would have to enter it, or it would have to be a hard-coded parameter to a program. Since our current system needs to verify only a clear-signed message by using a public key, PGP doesn't need a pass phrase. This makes the installation process easier and safer.

CERT's method for releasing software patches uses a similar system. They digitally sign e-mail messages and README files containing the checksums of files to be downloaded. People who take the time to verify the checksums can easily find out whether the files have been modified.

Weaknesses

This system has a couple of weaknesses. For one thing, it offers only file-level protection. It checks all the logical possibilities of modified files, deleted files and extra files. What about someone modifying the disk in some strange way that fools the upper level routines? Digitally signing a representation of the raw disk data is more secure.

This system is also vulnerable if the public key on the server could be modified or replaced with a different one. The same vulnerability exists for the upgrade software on the server. In practice, getting root access in order to replace the public key and creating unauthorized upgrades is a roundabout way to launch an attack. If the attacker already had root access on a particular computer, there wouldn't be any reason to use the upgrade system to get privileges or modify the server.

Authenticating Users without a Database

With the labs gradually coming on-line, we're dealing with the problem of authenticating student access. Most schools can just put a login program on their client PCs which checks the user's password against a central database via a high-speed campus LAN. This won't work for us for many reasons. Many of our labs will not be on-line in the near future. Even when on-line, the network support is unreliable and often slow. Another problem is that we have lots of weekend seminars students sign up for at the last minute. The students sign up for classes in small “education centers” that send us floppy disks with registration information via snail-mail long after the fact. Even if every lab were on-line, the logistics of collecting and distributing all of the information overnight would be extremely difficult. Luckily, public key cryptography and clear-signed digital signatures offer a solution:

1. A public and private key pair are generated for use by Field Representatives (FRs).
2. The FR public key is stored on the computer lab servers.
3. At registration time, a student entitled to computer lab access brings a floppy disk to the FR.
4. The FR clear-signs a certificate with the student's name, ID number, dates of validity and, optionally, information about which privileges are granted.
5. At least five characters are removed from the digital signature block and given to the student as his “password”. In Listing 3, the last five characters of the second encrypted line were used as the password.

1. The first time a student uses any particular computer lab, he inserts the disk into a client PC and enters his student ID number and password. The password and certificate on the disk are recombined and sent to the server where it is checked using the FR's public key. If the signature is both valid and unexpired (based on the dates in the certificate), access is granted.
2. One final step makes the system more convenient for students when returning to a lab. The server maintains a simple database, keyed on the student ID number with the student's password encrypted with a standard one-way encryption routine like **crypt(1)**. The next time a student visits that particular lab, he doesn't need to bring the floppy disk; he can just enter his password and be validated.
3. This database is automatically managed in the same way as a DNS cache. The date of expiration from the user's certificate becomes a “time to live” field in their database record. A **cron** job can be set up to periodically delete all expired entries.

This scheme has a lot of advantages. Reliable communication between the field staff, the main office and the labs is not required. Each of the three can be in completely separate, isolated locations. (And this is often the case.)

As in the first system, no real “secrets” are stored on the lab servers. If someone gets access to a lab server, there's no information that can help him: the public key can be read by anyone. The encrypted passwords are stored on the server; however, since they're random strings and not picked by the users, they aren't susceptible to the typical dictionary attack. Apart from dictionary attacks, standard Unix passwords are usually a secure system.

Various user access levels and periods of validity can be assigned to students by adding them to the certificates. The certificate can contain more information than just the student's name and ID number. Any kind of information that's worth keeping track of can be put into it.

Students have instant access to the computer lab once they've received their certificate. There is no delay waiting for a database to be propagated.

Since the certificates are in plain text, students can see at a glance if the certificates are correct, if they've expired yet, etc. This should make the system user-friendlier, and also limit the amount of assistance needed for help desk calls. There would never be any question about whether a certificate was still valid, issued for the correct person or contained the correct information.

Weaknesses

This scheme reduces the security requirements in the labs, but increases the precautions that need to be taken in the Field Staff's offices. Their PCs will have the secret keys on them, and plans should be made for when a computer is “compromised”. A possible fix is to give each education center a separate key set and put all possible public keys on the servers. If a system/key/passphrase is stolen, that particular key set is revoked and not used any more.

PGP and MD5 Resources

Robb Shecter is a longtime Unix user and has been a Linux fan since v. 0.98. He's interested in object-oriented design, Java, IP routing and bass guitar. He's currently a network and Unix specialist at the University of Maryland European Division, and can be reached at shecter@acm.org.

Archive Index Issue Table of Contents

Advanced search

Interview with Sameer Parekh

James T. Dennis

Issue #40, August 1997

Interview with Sameer Parekh of Stronghold.

Jim Dennis, “The Answer Guy” columnist for *Linux Gazette*, recently interviewed Sameer Parekh for *Linux Journal*. Sameer Parekh is the founder of C2Net Software Inc., http://www.c2.net/, the company that imports the Stronghold web server. Mr. Parekh has been active on the Internet since 1990, especially in areas related to privacy, electronic civil liberties, security and cryptography. Stronghold has added fully licensed commercial SSL support and other features to the popular Apache web server. Excerpts of Jim's interview are printed here; the full interview will be in the August 1 issue of *Linux Gazette*, http://www.ssc.com/lg/.

Jim: What can you tell me about C2Net as an organization? I know it used to be Community Connexions.

Sameer: Yes, we started out as an Internet provider and a privacy provider—protecting people's privacy on the Net. People could get anonymous accounts and set up anonymous web pages. We were strong supporters of the re-mailer network; we set up the means by which people can browse the net anonymously through our proxy. That business was going reasonably well. I was running it pretty much as a hobby in my spare time, while still a student at Berkeley. Then, I left school to start contracting at SGI down in the South Bay (the southern end of the San Francisco Bay area—near “silicon valley”.

At the end of last year we came out with Stronghold, though it wasn't called that at first. It was called “Apache-SSL U.S.” and started going really well. It became clear that we'd do a lot better selling cryptography products rather than privacy services. So, we moved our focus away from privacy services and changed our name to c2.net to reflect that change in focus and to concentrate primarily on deploying strong cryptography worldwide. As of a few months back, we officially had the name changed to C2Net Software Inc.

Jim: And you moved your customers over to Dave Sharnoff's idiom.com?

Sameer: We moved our dial-up customers over to idiom some time ago—around April, 1996. We were still supporting the privacy services until late last year, when we moved all our web hosting and anonymous account holders to Cyberpass, a company in San Diego. Cyberpass is run by a cypherpunk (a mailing list for the discussiong of the politics, technologies and social ramifications of cryptography and privacy issues) who is very active in privacy and in the re-mailer network.

Jim: You just mentioned the "cypherpunks"—you and I met at one of their meetings on the Stanford University campus in Palo Alto. Do you find most of your employees from this group?

Sameer: I find most of my employees from that group. The others I find from people I know from school, from other personal contacts and from existing employee referrals. Of the eleven employees I have, I met about half through cypherpunks.

I think a lot of my stand on privacy is related to my involvement with the cypherpunks and to my involvement in the controversy surrounding the clipper chip when it was first proposed.

We're the only company willing to deal with the fact that what the US government is trying to do with their export restrictions goes beyond just impeding or restricting export—it is creating a chilling effect so that companies inside the US would cripple their cryptography even for their domestic products. All of our development happens overseas so that we can do cryptography worldwide, and so that the international versions of our products will not have to be crippled to 40-bit keys that can be broken in three and a half hours.

Jim: So your approach is similar to the one John Gilmore and Hugh Daniels are using with the Free S/WAN project—keeping the developers at the other end, while you're providing the quality assurance on this side.

Sameer: Well, actually, we're providing mostly the marketing and the sales. We do a little bit of QA, but that's too close to the export issue. We also do the documentation—that's all written in the U.S. Also, all the protocols and the standardization efforts take place in the U.S. Stronghold conforms to protocols developed and published by Netscape, the W3 consortium and the IETF—among others.

Jim: Now, there's something I'm curious about. You've combined Apache and SSLeay, Eric A. Young's SSL (secure sockets layer) implementation, and integrated them into Stronghold. Then, you got a license from RSA so that you could include their public key libraries. How did you approach the Apache organization with the idea for a commercial version of their free package?

Sameer: Well, Apache is free under the Berkeley-style license, as opposed to the GPL, which means that I didn't have to have any connection to the Apache group. Apache's license allows you, so long as you insert the appropriate copyright notices, to start selling an Apache-based product without other delays.

But that would be kind of rude, I think. I'd been involved in the group before having any intention of changing the focus of my business. I saw a need for an SSL version of Apache that could be available within the U.S., so I started working on it and found SSLeay and Ben Laurie's Apache-SSL patches, which he'd done in the UK. I integrated these for limited distribution within the U.S.

I already knew many of the Bay Area members socially. I became a contributor to the Apache group—although not as big as the people who do large chunks of the code—I do testing and help with the documentation. A full-time tech writer employed at C2Net does documentation which she contributes back to the Apache group.

Our product is doing well, so our connection to the Apache group has been mutually beneficial. Any bug reports we get from our customers go back to them; any bugs we find, we fix and donate back. A large number of the features we've added, we've also donated back. Naturally, we haven't donated *all* of our added features, since we need to maintain some proprietary value so that we can make some money, as well.

Jim: What do you think about the GPL versus Berkeley issue? I know this is an ongoing bone of contention between the FreeBSD and Linux camps.

Sameer: I'm generally in favor of Berkeley over GPL because I think free software is best done in a variety of different contexts. In particular, with the crypto environment, it's impossible to do completely free software inside the U.S. if it involves any public key techniques, because of the patents. RSA holds a suite of patents which cover almost all known forms of public key encryption—patents are quite different from copyrights in that a re-implementation of the same algorithm is still covered.

I think the fewer the restrictions we place on our software, the more people will use it.

Jim: How many international programmers do you have and where are they located?

Sameer: We have two, and we don't say where they are. We don't want the U.S. government to know which country they're in, or they might pressure those countries to add export restrictions to their laws.

The NSA (National Security Agency) has appointed a person, David Aaron, whose sole job is to convince other countries to adopt restrictions similar to ours—so that our strategy won't continue to work. Obviously, if all other countries had similar export restrictions, doing development in any given one would allow sales only in that locale. That would be pointless in a global economy. We want to ensure that the country where we are doing our development will not be targeted.

Jim: Have you or any other company had any official contact with the NSA?

Sameer: I haven't, but I've heard a lot of rumors from companies who've had visits from them and were told, "What you're doing is wrong; you should stop it or it will do bad things to the rest of your business." NSA can't do that to me because I have no other business. We do cryptography, and we're at odds with export restrictions on intellectual property.

Jim: Would you see that as your edge against Microsoft, Netscape and Sun—that they would have other aspects of their business that might get severely hampered by the fight against cryptography export restrictions?

Sameer: Well, it's not worth it to them. It doesn't make good business sense for them. At the same time, it is a business necessity for us.

Any company who doesn't want to fight this battle can let us have that chore. They (and their offshore distribution agents) can license our software, and they won't have to do any development. They won't have to put their business at risk over questions of cryptography technologies.

Jim: How many platforms have you ported Stronghold to?

Sameer: Stronghold supports almost 20 different forms of Unix, including Linux. It supports both the ELF and the a.out libraries. It works with versions 1.2 and 2.0, although we recommend using the latest stable kernel.

Jim: Which implementation do you think is your biggest volume seller? They're all priced the same, right?

Sameer: Yes, they're all priced the same. Linux is our number one seller—next to it, we have Solaris and Irix. I haven't actually done the numbers because we don't sell on a "per-platform basis". We sell a Stronghold license, and the buyer can use it on whatever platform he wishes.

Jim: Now you have separate numbers for evaluation copies acquired and for copies licensed. About how many evaluation copies are being downloaded every month?

Sameer: On the order of 20 to 30 a day, which would come to about a couple hundred per week, or about 1,000 per month.

Netcraft shows that we have an installed base of about 20,000 on the public Internet, but that includes the virtual hosts, so it's not 20,000 actual hosts—it's the number of domains served by a Stronghold server. It's sort of a misleading number because they're checking only the non-SSL sites, and a lot of people run Apache on their unencrypted server and Stronghold on the encrypted server. Many run Stronghold on both, as well.

Netcraft did a different survey of SSL servers where we came in second. That is, for servers in general, we came out second among commercial Unix servers and fourth in commercial overall.

Jim: The Netcraft surveys you've been referring to, is there a link to them somewhere on your web pages?

Sameer: Well, it's at www.netcraft.com. I think the surveys are on our site as well. I'm proud of our Netcraft ratings, so we mention them prominently.

Jim: What other products are you working on?

Sameer: We have our "Safe Passage" web proxy. This does full-strength SSL for web browsers worldwide. It's currently in beta and is available at our U.K. site. It provides a locally-hosted proxy to provide full-strength cryptographic capabilities to the international versions of Netscape and Microsoft browsers. As you know, these are limited to 40-bit crypto when sold outside the U.S.—denying them access to sites requiring the stronger keys domestically available.

Basically, "Safe Passage" allows a user's browser to talk 40 bit to the proxy on his system which, in turn, talks to hosts out on the Web. Currently, it runs only under Windows.

Jim: What do you think of the Free S/WAN project?

S/WAN is a “secure wide-area networking” protocol from RSA—Free S/WAN is a work in progress which is being imported by another group of cypherpunks and John Gilmore of the EFF.

Sameer: I think it's a good thing. We need to provide IP level encryption in addition to the applications-specific security provided by programs like Stronghold or PGP. With regard to our product line, we haven't evaluated how that might fit into our strategy, so I don't have any comment from a business perspective.

However, from a more personal point of view, I think producing a freely available implementation of IP level encryption is a great thing. We want this deployed so that all of the Internet traffic is encrypted and, thus, authenticated.

Jim: Getting back to Stronghold as a “commercially supported Apache Server” and leaving aside its support for SSL and commerce... are there any companies offering just that—just a commercially packaged Apache?

Sameer: There are companies that offer Apache support services—but there aren't any selling a supported package—where you'd get a shrink-wrapped box, with binaries and pre-printed documentation, so these companies just offer the service. We offer a product—which includes e-mail support, of course.

Cygnus was doing some Apache support as well, but I believe they may have dropped that. Then there was a company in South Africa, Thawte, which had a product called Sioux. We ended up buying that one out and integrating its features with Stronghold's.

Sioux was released a few months after we had produced “Apache SSL U.S.” We started talking to Thawte—and decided to buy Sioux from them to eliminate any conflict of interest for some other business we wanted to do with them. You see, Thawte's primary business is as a CA (certification authority). It was an amicable arrangement, since Sioux wasn't the kind of software business they wanted to get into. We are now bundling Thawte certificates in the Stronghold package for only $50 US more—which is about half the regular price of a Thawte.

Jim: I've been reading in the Apache's modules lists about phps; what are they?

Sameer: php originally stood for “Personal Home Page”—but it doesn't mean that any more, so it's just php and doesn't stand for anything.

A php is a specific module that does dynamic content—a phrase I like to use for things like server side includes, extended ssi, php, e-perl, etc. They are all providing dynamic content—where the page is parsed by the server and the

data which are sent to the client are based on the scripting inside the original document.

We like php because it's easy to use, it's very robust and it offers connectivity to almost every database out there. Well, I shouldn't say that—there are a lot of databases “out there”. It can connect to Postgres '95, mSQL, Solid, Sybase, ODBC, etc.

It's a way to embed scripting inside your HTML. For example, you can have conditional sections that include blocks of HTML based on the results of certain pieces of code. You can have an HTML page that does a database query, formats and sends information out of the database. php offers significant speed advantages over CGI since it's loaded directly into the web server. You save the load of forking off a Perl process, which you must do when using CGI.

Stronghold 2.0 bundles with the php module—2.0 is in beta now. We've been using php quite a bit in-house for our database connectivity and our external web site.

We also support the server side includes—which were in the early CERN server. Stronghold is based on Apache, which also includes the “extended SSI”. XSSI adds things like conditionals.

Jim: Do you think tools such as these are better than CGI?

Sameer: Yes, it's a lot easier to build applications—particularly where it's not a complicated application—where you just want to include a little scripting directly in your HTML. If you use a CGI script, the script has to output all of the HTML. It's just as transparent to the browser—but it's a lot faster, and it's a lot easier for the web administrator to maintain.

Jim: Currently, the whole SSL view of the world, brought to us by the Netscape Commerce Server, is all about the server authenticating itself to the client—about web sites saying “You've reached me—and not some imposter and there's no *man-in-the-middle* and we can exchange information privately”. This approach doesn't seem to offer anything other than manually typed passwords. Maybe we need some sort of client authentication certificates for SSL.

Sameer: Actually, that's already in there. Stronghold already supports client authentication. The SSL protocol added that in version 2. Netscape supports client certificate authentication starting with Navigator version 3, which was built around the same time as SSL version 3.

Stronghold was the first widely-used commercial server to support SSL client authentication. Now that we have the support in the browser and our server, it's only a question of user acceptance and getting sites to start using it.

The SSL client authentication is an excellent technology. We're using it extensively here at C2Net. Because we have people from all over the world, we can't have this big private WAN, and we can't set up a VPN6 using something like Free S/WAN—it isn't ready yet.

Jim: Do you see C2Net coming out with, maybe, an ssltelnet and sslftp to compete with ssh?

Sameer: Well, we can't talk about all the details of all our product ideas. Actually, ssltelnet and sslftp already exist—no one's supporting them, and no one's using them yet.

I think for encrypting secure shell logins and file transfers, ssh is the best product out there. Although it's a different authentication protocol, unlike the SSL between my browser and my web server—it is RSA-based, and I can use my copy of ssh through my Ricochet and log in to my servers.

Jim: What else can you think of that just **has** to be said?

Sameer: The key thing that we at C2Net are focusing on is the worldwide deployment of cryptography. I think it's vital that we deploy strong crypto worldwide in the very near future.

The U.S. government has made it clear that their intent is to make the personal use of strong cryptography completely illegal. Deployment has to happen before they do that. If these crypto products aren't ubiquitous before then, we'll have a much harder time protecting our privacy.

I see cryptography being used for much more interesting things than just protecting credit cards. While it's prudent to encrypt your credit card number before sending it over the Net, it's not an interesting application of strong cryptography.

We want to build an infrastructure so that restrictions on personal use of privacy technology will have major business implications ... so that privacy itself cannot be made illegal.

Jim Dennis is the proprietor of Starshine Technical Services (http://www.starshine.org). His professional experience includes work in technical support, quality assurance and information services for both large and small software companies. He has just begun collaborating on the 2nd Edition of a book on Unix Systems Administration. Jim is an avid science fiction fan. He can be reached via e-mail at info@mail.starshine.org.

Archive Index Issue Table of Contents

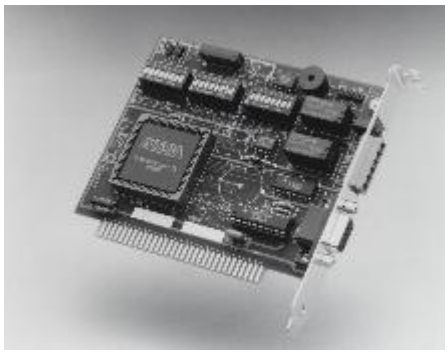
Advanced search

Berkshire PC Watchdog

David Walker

Issue #40, August 1997

The board can monitor a PC's activity in several ways to determine if it has locked up.



- Product: PC Watchdog
- Manufacturer: Berkshire Products
- Phone/Fax: 770-271-0088/770-271-0082x
- URL: http://www.berkprod.com/
- Price: $144.95 US $159.95 US with temperature monitor option
- Reviewer: David Walker

Do you have an Internet server that needs to be on-line 24 hours a day, 7 days a week dependably? A hardware watchdog timer is one way to be sure such a system is down for a minimal length of time. One such board is the PC Watchdog System Monitoring Board made by Berkshire Products.

I reviewed the PC Watchdog (rev. C) with the temperature monitoring option, part number 1090-1. From the manual: “The PC Watchdog board is a short, 8-bit ISA card that is used to monitor a PC to ensure maximum system availability.”

The board can monitor a PC's activity in several ways to determine if it has locked up. Dip switches on the board can be set to monitor specific I/O addresses for activity. If the PC Watchdog board does not detect activity on the monitored addresses for the specified period of time, it reboots the machine.

The board has a user I/O port that can be used for enhanced watchdog control and monitoring. This is the same interface used by the Linux kernel PC watchdog driver and PC watchdog daemon. If an I/O port on the board is not written to within the specified time, the board reboots the machine.

The board came packed in an anti-static bag in a box with a manual and a 3.5-inch MS-DOS disk of MS-DOS software, including source code. The manual covers the details of the hardware thoroughly. However, it did not specifically describe a Linux installation, and no Linux software is included on the disk.

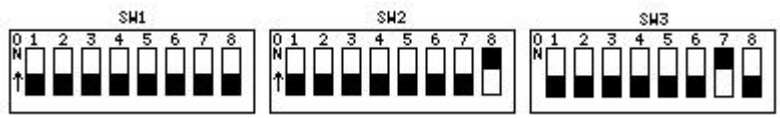
Platforms

The PC Watchdog comes with software drivers for MS-DOS/MS Windows. Linux support is available with the kernel and on the Internet. The board works with Intel architecture motherboards and requires one ISA slot.

Setup and Installation

The board uses three dip switches to configure its operation. I configured the board in order to ignore I/O activity as the Linux driver writes to the user I/O port to keep the board from resetting the PC. I set the address of the user I/O port to 0x0270 and set the delay time to one minute. My switch settings are shown in Figure 1.

Figure 1. Board Switch Settings



I compiled the Linux 2.0.28 kernel with the PC Watchdog driver enabled as a module. I also compiled the watchdog daemon from watchdog_2.0-0.tar.gz (from sunsite.unc.edu in /pub/Linux/system/Admin) and added it to /etc/rc.d/rc.local. I created /dev/watchdog and /dev/temperature with the major and minor device numbers specified in the kernel documentation on the watchdog (linux/Documentation/watchdog.txt).

When all was ready, I shut down my machine, turned off the power and installed the PC Watchdog board in an ISA slot, following the instructions in the manual.

A wire on the board connects to the reset connecter on the motherboard. The wire from the reset switch connects to another connecter on the Watchdog board, so that the reset switch on the case will still work.

Making It Work

When I turned the power on, my machine booted. After a 3.5-minute delay, the PC Watchdog beeped then rebooted my machine. After a few reboots I disconnected the wire from the board to the reset connector until I could figure out how to make the software work correctly.

I sent e-mail to Berkshire Products (73201.1270@compuserve.com) for any information they might have on Linux. Simon Machell promptly replied referring me to Ken Hollis (khollis@bitgate.com) who wrote the kernel driver for the PC Watchdog board.

While I waited to hear from Ken, I found a bug in the kernel driver. After I fixed this bug, the example watchdog daemon from linux/Documentation/watchdog.txt and the daemon from watchdog_2.0-0.tar.gz worked.

Listing 1 is my patch to fix the kernel driver included with Linux-2.0.28. It may also work with other kernels—your mileage may vary.

Ken directed me to the latest driver he has written: ftp://ftp.bitgate.com/pub/mirrors/bitgate/pcwd/pcwd-1.01.tar.gz. I got the tar file, looked at the contents, then patched my kernel source tree with the patch file patch-2.0.15.

Patching linux/drivers/char/pcwd.c and linux/include/linux/pcwd.h wasn't successful, so I copied pcwd-2.0.27.c to linux/drivers/char/pcwd.c and pcwd.h to linux/include/linux/pcwd.h. The watchdog driver then compiled successfully.

The new driver does not work with the daemons for the older driver; it comes with a new daemon. The driver works correctly with the included daemon. The daemon included with the driver lacks one useful feature: the daemon from watchdog_2.0-0.tar.gz. It doesn't fork when it writes to /dev/watchdog, so it won't reboot the machine if the process table gets full.

I modified the daemon to fork before writing to /dev/watchdog, so a full process table will cause a reboot of the machine. Listing 2 is the patch to watchdog.c from pcwd-1.01.tar.gz.

I did try compiling the PC Watchdog driver as part of the kernel, but it caused an error and wasn't initialized properly. It works fine compiled as a module.

Setting Up the System *rc* Files

The module must be loaded, and the watchdog daemon started before the file systems are fsck-ed. Fsck-ing the file systems can take longer than the delay built into the Watchdog board. I put the following commands to load the module and start the daemon in my /etc/rc.d/rc.S file (Slackware initialization files) before the file systems are checked.

```
# load the watchdog module and
# start the watchdog daemon
if [ -x /lib/modules/2.0.28/misc/pcwd.o ]; then
  echo -loading watchdog module'
  /sbin/insmod 'f /lib/modules/2.0.28/misc/pcwd.o
if [ -x /usr/sbin/watchdog ]; then
  echo -starting watchdog daemon-
  /usr/sbin/watchdog -t 10 &
fi
```

At this time, the root file system is mounted read-only so *depmod* cannot be run to build the modules.dep file. Therefore, *kerneld* won't be able to load the watchdog module when a new kernel is installed.

A generic link to the module directory can't be made at this time either; therefore, the full path name to the module must be used here. The path to the module must be updated when a new kernel is installed to insure that an old module is not loaded.

Testing

I tested the board by killing the watchdog daemon and running a program that forked until the process table was full. The system did not experience any failures on its own during testing.

The PC Watchdog can also monitor the temperature of the machine, although the kernel driver does not support reading the temperature. I wrote a short program to read and print the temperature reported by the board (see Listing 3). As I heated the board with a hair dryer, my program reported the rising temperature and the board started beeping an alarm when the temperature reached 56 degrees Celsius. The board does have an option to hold the PC in a reset state when the temperature exceeds 60 degrees Celsius by closing a relay. A daemon could be written to send e-mail or call a pager when the temperature gets too high or to shut down the PC.

Comparison with Other Products

Industrial Computer Source makes the WDT Watchdog Timer Hardware board, for which there is also a Linux kernel driver. It's available from Industrial Computer Source (619-677-0877 in the USA, 01-243-533900 in the UK, and (1)

69.18.74.30 France). It appears similar to the PC Watchdog board, though I've not used it.

A software watchdog driver is also available for the Linux kernel. The software watchdog cannot reboot the system from some lockups nor does it have a temperature sensor. The hardware boards should reboot the system after any lockup.

Conclusion

The PC Watchdog is a well-designed, well-made board. During my three weeks of testing, it operated dependably. The board never reset the PC unnecessarily, and it never failed to reset the machine when needed.

Berkshire Products

David Walker is Linux/Unix System Administrator and Programmer living near Seattle, Washington. When he isn't working he likes to play with Linux, hike or ride horses in the mountains. He can be reached at dwalker@eskimo.com.

Archive Index Issue Table of Contents

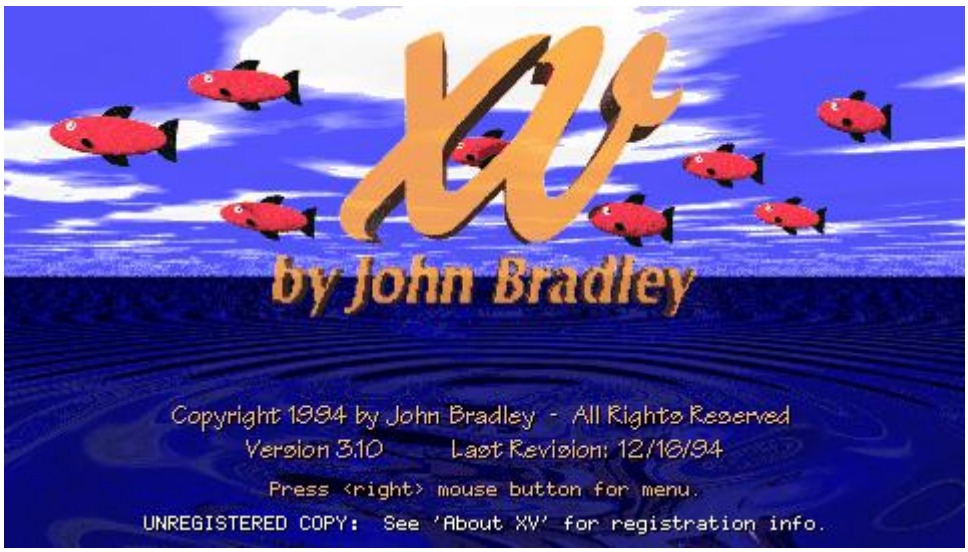
Advanced search

XVScan

Michael Montoure

Issue #40, August 1997

I did a search on DejaNews (http://www.dejanews.com/) while researching this review, to see if I could find any negative comments people had posted to Usenet about XVScan. I couldn't find any.



- Manufacturer: tummy.com, ltd.
- Phone: 970-223-8215; Fax: 408-490-2728
- E-Mail/URL: xvscan@tummy.com
- URL: http://www.tummy.com/xvscan/
- Price: $50 US for ftp or e-mail shippingAdditional $15 US for media in the United States$25 US internationally
- Platforms: Linux, HP-UX, BSD/OS, FreeBSD, SunOS and Solaris
- Reviewer: Michael Montoure

I'm a little biased—I had been happily using John Bradley's xv image manipulation software long before I ever heard of XVScan. So when I heard that someone had added the ability to acquire images from an HP ScanJet scanner

to xv, I was immediately intrigued. Sean Reifschneider, who wrote the software, posted the following message to comp.os.linux.hardware in 1995:

> When I first got the ScanJet I wrote a **hpscanpbm** (I don't think the real one was available then, and anyway, it only took 4 hours) that I used for a couple of months until I could get the time to write something better.

> The end result is "XVScan", a scanning extension to XV. While the "hpscanpbm" worked okay, this is about a thousand times better. I mean, I used to do the scan, load it into xv, and maybe I'd have to tweak, re-load, etc... Now I can just scan directly into XV.

Now, if you've never used xv, you might not understand my enthusiasm for it; it is, after all, a simple, straightforward tool. It certainly isn't in the same league as PhotoShop, but it's good at what it does—reading and writing files in a dozen different formats, window capturing, color-map editing, cropping and some fairly interesting image manipulation algorithms. In short, while it may not have all the fancy bells and whistles, it does have those things I use on a regular basis when manipulating images.

Installation

XVScan should run with any version of Linux—it's been tested with the 1.2.x and 2.0.x kernels, but it hasn't been tested with MkLinux yet. (Mac users—if you try this and get it to work, let tummy.com know—they're interested.) XVScan requires built-in generic SCSI driver support—no earlier than version 1.1.79. You don't need Motif, and you can use any version of XFree (X11R5, X11R6).

If you're not running Linux, you can also run XVScan under HP-UX, BSD/OS, FreeBSD, SunOS and Solaris—although if you're using Solaris and require SG, the generic pass-through SCSI driver, there is an extra charge.

XVScan was installed by Peter Struijk, one of SSC's Systems Administrators, before I started using it. A few minutes of looking through the documentation for XVScan makes installation seem easy, as Peter confirmed.

There's a setup program, used by the INSTALL-xvscan script, that searches your hardware for a scanner—namely, a SCSI ScanJet scanner, the only type XVScan can currently use. If one is found, the script creates the /dev/scanjet device file. Seems straightforward to me. In the Linux version, if XVScan can't find a scanner attached to the SCSI chain, you can still use the regular xv functions—just the scanning is disabled. Peter tells me the only tricky business is to remember that if in the future you change any of your SCSI devices, you must

rerun this setup program; otherwise, XVScan will no longer be able to find the scanner.

Some users, apparently, even pick up XVScan just to have an easy-to-install, pre-built copy of xv. This makes sense when you consider that the $50 price tag includes the $25 xv license fee, it's distributed with full source code and updates are free for the first year.

Getting Started

When I first started up XVScan, my first thought was, “This looks exactly like xv”--and it does. Only when you look at the Control menu do you notice the addition of a **Scanner** button that, when clicked, brings up the scanning window in which the scanned image is displayed. Other differences are even less apparent. Another change from the normal release of xv are the defaults used when XVScan starts.

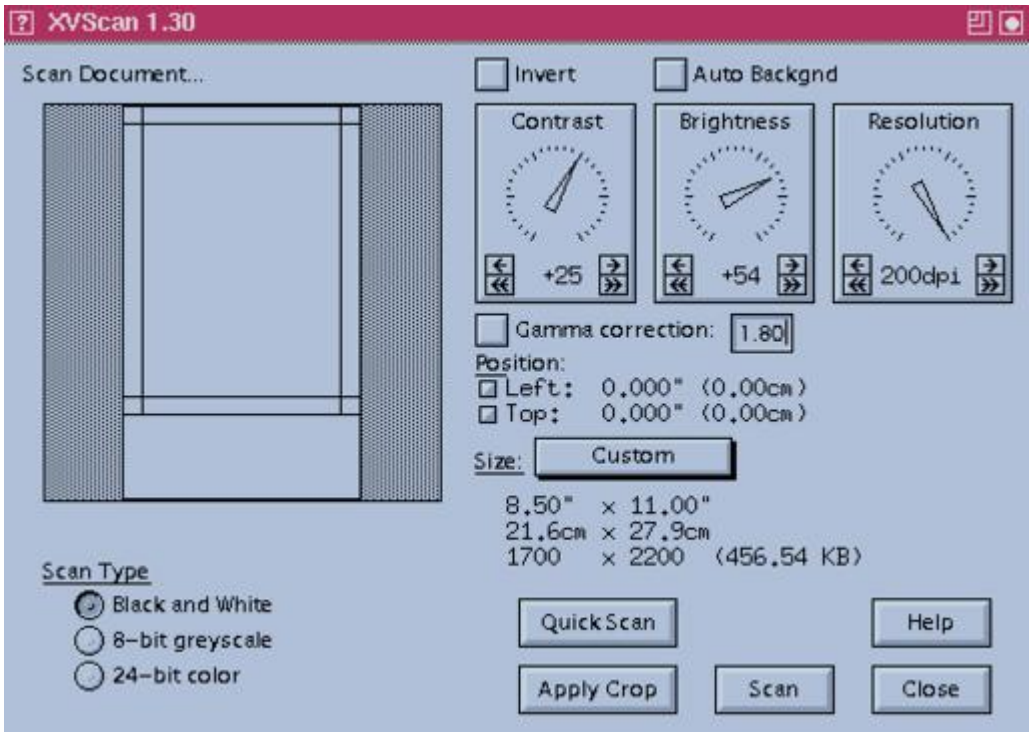
By default, XVScan turns on **-nolimits**, which lets images be larger than your screen—sometimes much, much larger. I suppose I can see the point of doing this, but I found it annoying not to have the image appear as a small, convenient window I could easily move around on my screen.

On the other hand, the other default it changes is that **-rwcolor**--read/write color entries—is set “on” at startup. Thus, any color editing I do to the image happens in real time, without having to manually **Apply** changes. That's kind of nice—I like it.

According to the documentation, both of these default options can be disabled. The documentation is handled nicely. When you click on the scanning window's **Help** button, it automatically launches a web browser that points at the on-line documentation at http://www.tummy.com/xvscan/. Therefore, you never have to remember their URL, and you don't have to keep a copy of their documentation on your hard drive. Of course, if you don't have an Internet connection up all the time, you might not find this so convenient.

The scanning window can be a little daunting to a first-time user, as you're immediately presented with a display of options, controls and buttons (see Figure 1). I suppose that's unavoidable for a program this flexible, but at first I was a little worried about the program's apparent complexity.

Figure 1. XVScan Control Window



I shouldn't have worried. The interface is surprisingly easy to decipher and use. There are generally several ways to do any one task. For example, if you want to resize the area you're scanning, you can enter numbers to resize it, pick from a standard selection of paper sizes (conveniently, you can define your own common document sizes) or click and drag the bars at the edges of the scan window.

Scanning

Scanning is fast and the quality is high; XVScan can quickly and easily handle scanning all the way up to 600 dpi. You can precisely select the dpi to use for scanning and you can adjust the brightness, the contrast, the gamma correction—most anything you'd like to control.

Unlike a lot of scanners, XVScan doesn't give you a miniature of the image in the “pre-scan” mode. Instead, when the **QuickScan** option is selected, XVScan scans the image and adjusts the resolution so that the resultant image exactly fits the scanning window. This action might end up displaying the image at a lower resolution than you have in mind for your final image; it might just as easily be higher. (For me, it was often higher. I scan images to look good on a web page, not to be printed, so I don't need to scan at resolutions higher than 72 dpi.)

Once you have your QuickScan image on screen, you can adjust the appropriate settings as you wish to enhance it and, if necessary, set crop limits. Once you click on **Apply Crop**, the cropped image is displayed filling the scanning window. Now, click on **Scan** for your next pass, and XVScan scans only the area inside the crop limits—very convenient.

That's great, I can hear you say, but I don't always want to play around in a fancy X Windows' user interface—sometimes I'd like to be able to just scan an image using the command line. You can do that, too. Although their manual boasts that the GUI makes their product "more user friendly than **hpscanpbm**", it's also distributed with **sjscan**, a command-line scanning utility.

By the way, while XVScan can read and write in several different image formats, their manual warns:

> xv has a whopping grand total of two internal image formats: 8-bit color mapped and 24-bit RGB. Every image you load is converted to one of these two formats, as part of the image loading procedure, before you ever see the image.

In other words, you might occasionally get some minor artifacts creeping into the images as they're converted back and forth—but if this is a serious problem, I have yet to encounter it.

Conclusion

I did a search on DejaNews (http://www.dejanews.com/) while researching this review, to see if I could find any negative comments people had posted to Usenet about XVScan. I couldn't find any. The closest thing I found to a negative remark was a few die-hard free software fans grumbling about having to actually pay money for it. On the whole, everyone who had used the software seemed to recommend it. In fact, in response to a general question about scanning under Linux, one user suggested that the person asking should buy an HP ScanJet just so he could run XVScan.

Personally, I like having an environment that allows me to do high-quality scans and to manipulate the image within the same application. I'm far too used to programs that do just one thing well; XVScan appears to aim at doing everything well, and I think it succeeds.

Michael Montoure is the webmaster for Specialized Systems Consultants, the publishers of *Linux Journal*. He's been on the Internet for the better part of a decade now, used to wish everyone would use it, and now is sorry he said anything. As long as you don't want to tell him how to "Make Money Fast", feel free to send him e-mail at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

The Java Series

Kirk Petersen

Issue #40, August 1997

Since The Java Series is from the writers of the actual software, it is the official source of information regarding the Java programming language, virtual machine and API.

- Publisher: Addison-Wesley
- Phone/Fax: 617-955-3700; 617-942-3077
- URL: http://www.aw.com/devpress/
- Reviewer: Kirk Petersen

If you have recently visited the computer section of a bookstore, you have seen the mind-boggling number of Java-related books available. Many bookstores have sections dedicated solely to Java, just as they have sections for programming languages and the Internet. What makes this even more incredible is that the first Java book showed up just a little over a year ago.

The Java Series is a set of nine books written by members of the Java team. This series is published by Addison-Wesley.

Since *The Java Series* is from the writers of the actual software, it is the official source of information regarding the Java programming language, virtual machine and API. If you want the definitive answer to a Java-related question, these books are the place to look.

This review covers five of the books from *The Java Series*. The first four are *The Java Virtual Machine Specification*, *The Java Language Specification* and *The Java Application Programming Interface* (two volumes). These are the technical references that describe the entire Java environment. Also included is a review

of *Concurrent Programming in Java*, which isn't a definitive reference like the rest but certainly deserves its place in the series.

- Title: The Java Virtual Machine Specification
- Authors: Tim Lindholm and Frank Yellin
- Price: $36.75
- ISBN: 0-201-63452-X

All too often, people either forget or don't know that Java is more than just a programming language. Actually, the most important part of Java may be the virtual machine. It provides the binary portability that makes Java so important in network environments, and where would Java be without the Internet? That is why I chose to start with *The Java Virtual Machine Specification*.

One way of looking at the JVM is to think of it as just another 32-bit CPU. It has many of the same components, including an instruction set, registers and the ability to do common operations on a similar set of data types. In many ways, *The Java Virtual Machine Specification* is like a reference manual you would get from a major CPU manufacturer—it contains an instruction reference and other reference material.

However, just as the JVM isn't exactly like most CPUs, *The Java Virtual Machine Specification* isn't exactly like most CPU reference manuals. It is a well-written book that is more than just instruction set references and explanations of the data types. This book (excluding the instruction set reference) can be read straight through and enjoyed. For example, Chapter 7 provides a very interesting tutorial called “Compiling for the JVM”. Not only would this be useful to compiler writers, but it would give any Java programmer insight into what his code is really doing. Actually, the book presents this information so well that it might encourage people, who wouldn't otherwise do so, to write useful bytecode utilities.

The book begins with a short and unexceptional introduction. It consists of a one-page history of Java, a one-page introduction to the virtual machine and a very brief chapter summary.

Chapter 2 is titled “Java Concepts”, and it summarizes the features of the Java programming language. The next chapter details the structure of the Java Virtual Machine. Data types, registers, stacks and many similar concepts are defined. The class file format and instruction set are introduced towards the end of Chapter 3.

In Chapter 4, the class file format is defined. Like most file format descriptions, a pseudo-code data structure is given which shows how the data are arranged

in the file. Each field in this structure is described in an easy-to-understand and thorough manner.

Constant pool resolution is the topic of Chapter 5. The constant pool is similar to the symbol table found in conventional languages. In Java the constant pool allows the instruction set to refer to objects with a pair of bytes instead of the full name. In addition to describing the reason for the constant pool, the chapter explains how it is used in the process of loading, linking and initializing class files.

The instruction set for the Java Virtual Machine is found in Chapter 6. A short introduction is followed by 180 pages of well- organized bytecode descriptions.

Chapter 7 outlines the process of compiling source code to the JVM. This is done with examples of Java source code and the corresponding JVM bytecodes. The authors then describe how and why the compiler generated the bytecodes it did. Many examples in the chapter describe everything from compiling a simple **for** loop to throwing and handling exceptions.

The Java Virtual Machine Specification is a very useful book that is well worth reading. Although it is fundamentally a reference, it is remarkably easy to read. Therefore, I recommend it to anyone who wants to know more about the low-level workings of Java.

- Title: The Java Language Specification
- Authors: James Gosling, Bill Joy and Guy Steele
- Price: $36.75
- ISBN: 0-201-63451-1

The Java Language Specification defines the Java programming language. For those of you who are trying to learn the Java programming language, you should know that this book is not a tutorial. It is a reference that includes every possible detail of the Java programming language.

In addition to defining the structure of the language, *The Java Language Specification* has numerous code samples that help illustrate specific rules. These are very helpful, especially since the descriptions often favor syntactic precision over human understanding.

The introduction consists of a quick description of Java and a chapter summary. It is followed by a brief chapter describing the grammar the rest of the book uses to explain the Java language.

The lexical structure of Java is described in Chapter 3. It begins with some notes on how Java uses Unicode, then goes into the low-level parsing of the source code files. This includes the definition of things such as white space, identifiers, keywords and so on. The chapter ends with a definition of the various types of literals, separators and operators.

“Types, Values and Variables” (Chapter 4) defines the various types that are available in Java. Primitive types are discussed first, including the ranges and operations of the integer, floating-point and boolean types, then reference types are defined. Since references deal with objects, some of the more primitive objects are outlined, including the class **Object** and class **String**. The chapter ends with a summary of the seven different types of variables in Java.

“Conversions and Promotions” (Chapter 5) defines the ways that the type of a variable can be changed. Conversions on both primitive and reference types are defined, as is the unique string conversion.

Chapter 6 defines the concept of a name in Java. Names are used to refer to packages, classes, members of an object and variables. The concepts of scope, inheritance and access control are also defined to the extent that they are used in naming.

In Java, classes can be organized into groups based on their functionality. These groups, called packages, are defined in Chapter 7. Included in this chapter are conventions for the naming and storing of packages.

Classes and interfaces are described in Chapters 8 and 9. These chapters provide good descriptions of all aspects, from keywords that modify classes and interfaces to the workings of inheritance and overloading.

Chapter 10 describes how arrays work in Java. The chapter begins with the assertion that arrays are objects, but of a distinct type. The chapter describes how arrays are similar to and different from other objects.

Exceptions are defined in Chapter 11. The chapter begins with a basic description of how the language constructs interact with the **Throwable** objects. It moves on to the causes of exceptions, and the end of the chapter describes all the exceptions the programmer will face.

Chapter 12 defines in an interesting way how the language interacts with the Java Virtual Machine. It begins with a description of what the virtual machine does when it is starting up. It moves on to a description of the loading and linking of classes, and ends with a description of how the virtual machine exits.

Probably the most interesting chapter of the book is Chapter 13. It describes in what ways old class files should remain compatible with new class files. The authors give many examples of code that should be binary compatible and how to make sure they are.

Chapters 14 and 15 describe the more traditional aspects of the Java language. The statements are defined, and then the way in which they are executed is outlined.

In Java, a value must be assigned to a local variable before it can be used. This is called “definite assignment” and is the topic of Chapter 16. It is very slow reading and will most likely be best used as a reference.

Threads and locks are introduced in Chapter 17, beginning with a description of how things differ when more than one thread of Java code is executing at a time. The chapter ends with some examples of situations that make multi-threaded code difficult, and the corresponding Java constructs that can solve the problems.

Java takes the two comment styles present in C++ and adds a third. This new comment style allows class interfaces, hierarchies and programmer documentation to be turned into web pages. Chapter 18 gives a description and an example.

Chapter 19 lists the LALR(1) grammar—probably of tremendous interest to compiler writers.

Finally, the book ends with an API reference that covers the java.lang, java.util and java.io packages. These are the core packages that must be present in all Java environments. They are covered in a bit more detail here than they are in *The Java API*.

The Java Language Specification is a good language reference. However, I don't think most people need a book this specific. I recommend it primarily to compiler writers or anyone else who is building a language-based tool.

- Title: The Java Application Programming Interface, Volumes I & II
- Authors: James Gosling, Frank Yellin and The Java Team
- Price: $38.75 each
- ISBN: 0-201-63453-8, 0-201-63453-7

The Java API is a reference book describing the classes that are part of the Java Development Kit. It is fairly simple and doesn't contain very detailed descriptions.

The book is split into two volumes sold separately. Volume I contains “Class Hierarchy Diagrams”, “Package java.lang”, “Package java.io”, “Package java.util” and “Package java.net”. Volume II contains “Package java.awt”, “Package java.awt.image”, “Package java.awt.peer” and “Package java.applet”.

I have a hard time recommending *The Java API*. There are now several books that are better deals. *The Class Libraries* is basically a superset of *The Java API* with more information about the way the classes work, and it includes numerous code samples. Or, if you are looking to save some money, you might want to get *Java in a Nutshell* by O'Reilly and Associates. It has a very nice, albeit small, API reference and only costs $20. It is my favorite Java book and is nice and soft from nearly a year of constant use.

- Title: Concurrent Programming in Java
- Authors: Doug Lea
- Price: $23.86
- ISBN: 0-201-69581-2

Although there are many Java books out there, I haven't found many that use Java to teach an advanced programming concept. *Concurrent Programming in Java* is one of the few. I have always been aware of Java's concurrency mechanisms, but this book taught me how they can be used for real work.

The introduction is much better than those of the other four Java books reviewed here. First, the author lists some advantages and limitations of concurrency. Then the purpose of the book is given, followed by a typical chapter summary and a description of Java's concurrency mechanisms.

Chapter 2 defines the concept of safety with regard to concurrent programming; it outlines the methods by which multiple threads can execute on the same data so that “nothing bad ever happens”. These methods include using immutable objects, synchronization and containment.

Liveness could be considered the opposite of safety. Chapter 3 describes liveness problems and methods that can be used to solve them.

Chapter 4 covers the concept of state-dependent action. The author describes many possible actions that can be taken when a set of states is reached.

Concurrency control is covered in Chapter 5. Three approaches to concurrency control are covered. Examples are shown using Java, but the concepts are useful in any language.

Chapter 6 is titled “Services in Threads”. It explains how common tasks can be separated into threads. At this point the book becomes really interesting, because the reader can see how the concurrency concepts can work in real programs.

Chapter 7 describes how the flow policies implemented by a program affect safety and liveness issues. Finally, methods of coordinating more independent objects are outlined in Chapter 8.

Concurrent Programming in Java is a very good book. It is nice to see a book go into the details of a subject that hasn't already been beaten to death (the Applet class, for example).

If you have a lot of experience doing multi-threaded programming, you may find *Concurrent Programming in Java* a bit too simple and vague. Probably all you need is a description of Java's concurrency mechanisms. However, if you have little or no experience making concurrent programs, this book has a lot to teach you.

Conclusions

The series as a whole works well. As the marketing material says, it represents the definitive source of Java information. I haven't found any books outside this series that add much in terms of raw information.

Although these books are well-produced and provide valuable information, I can't recommend that everyone go buy them. If you are a student with a limited budget, as I am, you might want to look into something like *Java in a Nutshell* (O'Reilly and Associates). It provides a tutorial for C programmers who are trying to learn Java and also has an good API reference. Other places to find cheap information on Java include the SSC API reference cards and JavaSoft's web page.

On the other hand, if the price of a $30-$40 reference book doesn't make a dent in your wallet and you need to have the official source of Java information, all five of these books are very good deals.

One final consideration—a new version of the Java Development Kit has just been released. It has introduced numerous changes and additions to the class library. Therefore, I suggest waiting to buy an API reference until it includes JDK 1.1 information.

Kirk Petersen is a senior attending The Evergreen State College. He is currently searching for a job (Java and/or Linux) in Seattle. Leisure activities include playing pool and studying music. His e-mail address is kirk@speakeasy.org.

Archive Index Issue Table of Contents

Advanced search

The Linux Database

Sid Wentworth

Issue #40, August 1997

This volume covers a database that just happens to use Linux as the underlying platform, which makes it quite different from most Linux books.

- Author: Fred Butzen & Dorothy Forbes
- Publisher: MIS:Press
- ISBN: 1-55828-491-5
- Price: $39.95
- Pages: 561 (plus CD)
- Reviewer: Sid Wentworth

The Linux Database is the third in the *Slackware* series from MIS:Press.

This volume covers a database that just happens to use Linux as the underlying platform, which makes it quite different from most Linux books.

The first part of the book deals with relational databases and the second with programming a database application. Both sections contain good information from which much can be learned about databases. The word Linux does not appear within the first section.

There are three chapters in the first part. The first chapter introduces the relational modes, the second addresses the principles of database design and the final chapter introduces structured query language (SQL). The first chapter deals with theory and introduces the idea of a baseball team database whose development is followed throughout. Chapters 2 and 3 include examples of database issues which are easy for someone with a programming background to understand.

The first chapter in Part 2 introduces the architecture of database applications and presents the three-tier model: access, business logic and user interface.

The following chapters introduce the Open DataBase Connectivity (ODBC) call line interface, the Java DataBase Connectivity (JDBC) applications program interface and embedded SQL. The author explains how to actually access a database using each of these three methods. The focus is on interconnecting the database with a web page.

Chapter 6 develops the front ends using both HTML/CGI and Java. Examples of code and HTML increase at this point in the book. The final chapter presents what the authors call *middleware*, the semantic logic that controls the actions of the other two tiers.

Each chapter concludes with a summary and references to other sources of information, which are both current and the right choices. The CD includes a standard Slackware distribution of Linux plus four databases: Just Logic, mSQL, Postres95 and Ingres. The included version of Just Logic is “crippleware”--that is, it will process only 1000 commands. mSQL is free for non-commercial use, and information on registering for commercial use is included. Postgres and Ingres are both under the UCB (University of California Berkeley) copyright, which pretty much grants free use for any purpose.

The Linux Database is well written and fairly easy to read in spite of the fact that much of the material is rather technical in nature. If you are interested in relational databases, this book, the included CD, a PC and a lot of time can get you up to speed. If you aren't a student but have a professional interest in a relational database, the included tools and documentation will get you well on your way. Also, having four databases to work with will help you pick the right tool for your task.

Sid Wentworth lives in Uzbekistan, where he divides his time between UUCP hacking, raising yaks and visiting the tomb of his personal hero, Tamerlane the Great.

Archive Index Issue Table of Contents

Advanced search

A Web Crawler in Perl

Mike Thomas

Issue #40, August 1997

Here's how spiders search the Web collecting information for you.

Web-crawling robots, or spiders, have a certain mystique among Internet users. We all use search engines like Lycos and Infoseek to find resources on the Internet, and these engines use spiders to gather the information they present to us. Very few of us, however, actually use a spider program directly.

Spiders are network applications which traverse the Web, accumulating statistics about the content found. So how does a web spider work? The algorithm is straightforward:

1. Create a queue of URLs to be searched beginning with one or more known URLs.
2. Pull a URL out of the queue and fetch the Hypertext Markup Language (HTML) page which can be found at that location.
3. Scan the HTML page looking for new-found hyperlinks. Add the URLs for any hyperlinks found to the URL queue.
4. If there are URLs left in the queue, go to step 2.

Listing 1 is a program, spider.pl, which implements the above algorithm in Perl. This program should run on any Linux system with Perl version 4 or higher installed. Note that all code mentioned in this article assumes Perl is installed in /usr/bin/Perl. These scripts are available for download on my web page at http://www.javanet.com/~thomas/.

To run the spider at the shell prompt use the command:

```
spider.pl <starting-URL<search-phrase>
```

The spider will commence the search. The starting URL must be fully specified, or it may not parse correctly. The spider searches the initial page and all its

descendant pages for the given search phrase. The URL of any page with a match is printed. To print a list of URLs from the SSC site containing the phrase "*Linux Journal*", type:

```
spider.pl http://www.ssc.com/ "Linux Journal"
```

The Perl variable **$DEBUG**, defined in the first few lines of spider.pl, is used to control the amount of output the spider produces. **$DEBUG** can range from 0 (matching URLs are printed) to 2 (status of the program and dumps of internal data structures are output).

Interaction with the Internet

The most interesting thing about the spider program is the fact that it is a network program. The subroutine **get_http()** encapsulates all the network programming required to implement a spider; it does the "fetch" alluded to in step 2 of the above algorithm. This subroutine opens a socket to a server and uses the HTTP protocol to retrieve a page. If the server has a port number appended to it, this port is used to establish the connection; otherwise, the well-known port 80 is used.

Once a connection to the remote machine has been established, **get_http()** sends a string such as:

```
GET /index.html HTTP/1.0
```

This string is followed by two newline characters. This is a snippet of the Hypertext Transport Protocol (HTTP), the protocol on which the Web is based. This request asks the web server to which we are connected to send the contents of the file /index.html to us. **get_http()** then reads the socket until an end of file is encountered. Since HTTP is a connectionless protocol, this is the extent of the conversation. We submit a request, the web server sends a response and the connection is terminated.

The response from the web server consists of a header, as specified by the HTTP standard, and the HTML-tagged text making up the page. These two parts of the response are separated by a blank line. Running the spider at debug level 2 will display the HTTP headers for you as a page is fetched. The following is a typical response from a web server.

```
HTTP/1.0 200 OK
Date: Tue, 11 Feb 1997 21:54:05 GMT
Server: Apache/1.0.5
Content-type: text/html
Content-length: 79
Last-modified: Fri, 22 Nov 1996 10:11:48 GMT
<HTML><TITLE>My Web Page</TITLE>
<BODY>
This is my web page.
```

```
</BODY>
</HTML>
```

The spider program checks the **Content-type** field in the HTTP header as it arrives. If the content is of any MIME type other than text/html or text/plain, the download is aborted. This avoids the time-consuming download of things like .Z and .tar.gz files, which we don't wish to search. While most sites use the FTP protocol to transfer this type of file, more and more sites are using HTTP.

There is a hardware dependency in **get_http()** that you should be aware of if you are running Linux on a SPARC or Alpha. When building the network addresses for the socket, the Perl **pack()** routine is used to encode integer data. The line:

```
$sockaddr="S n a4 x8";
```

is suitable only for 32-bit CPUs. To get around this, see Mike Mull's article “Perl and Sockets” in *LJ* Issue 35.

The URL Queue

Once the spider has downloaded the HTML source for a web page, we can scan it for text matching the search phrase and notify the user if we find a match.

We can also find any hypertext links embedded in the page and use them as a starting point for a further search. This is exactly what the spider program does; it scans the HTML content for anchor tags of the form **<A HREF="url">** and adds any links it finds to its queue of URLs.

A hyperlink in an HTML page can be in one of several forms. Some of these must be combined with the URL of the page in which they're embedded to get a complete URL. This is done by the **fqURL()** function. It combines the URL of the current page and the URL of a hyperlink found in that page to produce a complete URL for the hyperlink.

For example, here are some links which might be found in a fictitious web page at http://www.ddd.com/clients/index.html, together with the resulting URL produced by **fqURL()**.

URL in Anchor Tag	Resulting URL
http://www.eee.org/index.html	http://www.eee.org/index.html
att.html	http://www.ddd.com/clients/att.html

/att.html	http://www.ddd.com/att.html

As these examples show, the spider can handle both a fully-specified URL and a URL with only a document name. When only a document name is given, it can be either a fully qualified path or a relative path. In addition, the spider can handle URLs with port numbers embedded, e.g., http://www.ddd.com:1234/index.html.

One function not implemented in **fqURL()** is the stripping of back-references (**../**) from a URL. Ideally, the URL /test/.../index.html is translated to /index.html, and we know that both point to the same document.

Once we have a fully-specified URL for a hyperlink, we can add it to our queue of URLs to be scanned. One concern that crops up is how to limit our search to a given subset of the Internet. An unrestricted search would end up downloading a good portion of the world-wide Internet content—not something we want to do to our compadres with whom we share network bandwidth. The approach spider.pl takes is to discard any URL that does not have the same host name as the beginning URL; thus, the spider is limited to a single host. We could also extend the program to specify a set of legal hosts, allowing a small group of servers to be searched for content.

Another issue that arises when handling the links we've found is how to prevent the spider from going in circles. Circular hyperlinks are very common on the Web. For example, page A has a link to page B, and page B has a link back to page A. If we point our spider at page A, it finds the link to B and checks it out. On B it finds a link to A and checks it out. This loop continues indefinitely. The easiest way to avoid getting trapped in a loop is to keep track of where the spider has been and ensure that it doesn't return. Step 2 in the algorithm shown at the beginning of this article suggests that we “pull a URL out of our queue” and visit it. The spider program doesn't remove the URL from the queue. Instead, it marks that URL as having been scanned. If the spider later finds a hyperlink to this URL, it can ignore it, knowing it has already visited the page. Our URL queue holds both visited and unvisited URLs.

The set of pages the spider has visited will grow steadily, and the set of pages it has yet to visit can grow and and shrink quickly, depending on the number of hyperlinks found in each page. If a large site is to be traversed you may need to store the URL queue in a database, rather than in memory as we've done here. The associative array that holds the URL queue, **%URLqueue**, could easily be linked to a GDBM database with the Perl 4 functions **dbmopen()** and **dbmclose()** or Perl 5 functions **tie()** and **untie()**.

Responsible Use

Note that you should not unleash this beast on the Internet at large, not only because of the bandwidth it consumes, but also because of Internet conventions. The document request the spider sends is a one line **GET** request. To strictly follow the HTTP protocol, it should also include **User-Agent** and **From** fields, giving the remote server the opportunity to deny our request and/or collect statistics.

This program also ignores the “robots.txt” convention that is used by administrators to deny access to robots. The file /robots.txt should be checked before any further scanning of a host. This file indicates if scanning from a robot is welcome and declares any subdirectories that are off-limits. A robots.txt file that excludes scanning of only 2 directories looks like this:

```
Useuagent: *
Disallow: /tmp/
Disallow: /cgi-bin/
```

A file that prohibits all scanning on a particular web server looks like this:

```
User-agent: *
Disallow: /
```

Robots like our spider can place a heavy load on a web server, and we don't wish to use it on servers that have been declared off-limits to robots by their administrators

Application of the spider.pl Script

How might we use the spider program, other than as a curiosity? One use for the program would be as a replacement for one of the web site index and query programs like Harvest (http://harvest.cs.colorado.edu/Harvest/) or Excite for Web Servers (http://www.excite.com/navigate/prodinfo.html). These programs are large and complicated. They often provide the functionality of the Perl spider program, a means of archiving the text retrieved and a CGI query engine to run against the resulting database. Ongoing maintenance is required, since the query engine runs against the database rather than against the actual site content; therefore, the database must be regenerated whenever a change is made to the content of the site.

Some search engines, such as Excite for Web Servers, cannot index the content at a remote site. These engines build their database from the files which make up the web site, rather than from data retrieved across a network. If you had two web sites whose content was to appear in a single search application, these tools would not be appropriate. Furthermore, the Linux version of Excite for Web Servers is still in the “coming soon” stage.

Listing 2 and Listing 3 show a simple CGI search engine that is implemented using the spider.pl program. Listing 2 is an HTML form which calls spiderfind.cgi to process its input. Listing 3 is spiderfind.cgi. It first uses Brigitte Jellinek's library to move the data entered in the form into an associative array. It then calls the spider.pl program using the Perl **system()** function and passes the form data as parameters. Finally, it converts the output from spider.pl into a series of HTML links. The user's browser will display a list of hyperlinked URLs in which the search text was found. Note that the name of the host to search is specified by a hidden field in the HTML document. There are better and more security-conscious ways for two Perl programs to interact than through a Perl **system()** call, but I wanted to use an unmodified copy of spider.pl for this demonstration.

This script doesn't provide the complete functionality of the packages mentioned above, and it won't perform as well. Since we're doing the search against web server documents across the Net, we don't have the advantage of index files; therefore, the search will be slower and more processor-intensive. However, this script is easy to install and easier to maintain than those engines.

Another application that could be built using the spider.pl program is a broken link scanner for the Web. The HTTP response we showed previously began with the line "**HTTP/1.0 200 OK**", indicating the request could be fulfilled. If we tried to hit a URL with a non-existent document, we would get the line "**HTTP/1.0 404 Not found**" instead. We could use this as an indication that the document does not exist and print the URL which referenced this page.

The modifications to the spider program needed to accomplish this are minor. Every time a hyperlink's URL is added to the URL queue, we also record the URL of the document in which we found the hyperlink. Then, when the spider checks out the hyperlink and receives a "**404 Not found**" response, it outputs the URL of the referring page.

Mike Thomas is an Internet application developer working for a consulting firm in Saskatchewan, Canada. Mike lives in Massachusetts and uses two Linux systems to telecommute 2000 miles to his job and to Graduate School at the University of Regina. He can be reached by e-mail at thomas@javanet.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Templates: Separating Programs from Design

Reuven M. Lerner

Issue #40, August 1997

Make web site design changes easier by using templates—HTML pages with embedded Perl code.

If you are running a small web site, then you are probably responsible for everything—server administration, the site's content and the CGI programs that produce variable and dynamic content.

If your site is of any significant size, the work is probably divided among a number of people. Indeed, most large sites divide their staffs between the people who are responsible for the site's content and design, and those who are responsible for the infrastructure and technical side of the site.

Such a division undoubtedly makes it easier to administer a site. After all, it is much easier to find someone to write content or to write CGI programs than to do both. In addition, splitting the work according to function allows everyone to do what he does best.

At the same time, such a division makes it difficult for sites to maintain a uniform presentation style. CGI programs produce HTML that must match the style of the rest of the site. This might mean inserting a certain header at the top of each page, using a particular background color or inserting a graphic on the side of each page.

In other words, there are two separate sources for HTML content on a web site. The pages of HTML created by the designers, and the HTML produced by the CGI programs. If a site does not change its style often (or at all), the fact that the HTML comes from two sources does not matter. The designers establish a style for the site, which is then adopted by both designers and programmers for their work.

However, many sites have taken to redesigning every few months, partly due to continually improved technology that allows designers to create more interesting, exciting experiences on their sites. Every time a site's design changes, all of the existing content must be rewritten to fit the new design. Sites that have split their content between programmer-generated HTML and CGI-generated HTML will find themselves having to convert two types of files with two separate staffs.

For example, let's assume that a site has standardized white-on-blue text. Each time the designers create a new page, they make sure to include a **<body>** tag of the form:

```
<body bgcolor="blue" fgcolor="white">
```

In order for the site to have a uniform look, all of the CGI programs on this site must include a similar **<body>** tag at the top of their output. Here is a basic "hello, world" page that demonstrates how to accomplish this:

```
#!/usr/bin/perl -w
use strict;
use diagnostics;
use CGI; # Available from http://www.perl.com/CPAN
# Create an instance of CGI
my $query = new CGI;
# Send an appropriate MIME header
print $query->header("text/html");
# Begin the HTML, with our colors indicated
print $query->start_html(
        -title => "Hello, world!",
         -bgcolor => "blue",
         -fgcolor => "white");
# Send our message
print "<P>Hello, world!</P>\n";
# End the HTML
print $query->end_html;
```

If this program were run as a CGI program from within a web server, it would produce a short page of HTML on our screens, with a the text appearing in white text on a blue background. (And yes, we should use hex codes for consistent colors across platforms, but this is just meant to be an easy example.)

After creating an instance of CGI (an object module freely available from CPAN at http://www.perl.com/CPAN), the program sends a MIME header indicating that it will be sending HTML-formatted text to the user's browser. Following that MIME header, it sends a **<body>** tag, hidden somewhat by the **start_html** method that takes care of such tag production for us.

Finally, we send our short message, marked up in HTML, and invoke the end_html method, which sends a **</body>** tag to end the body of the HTML text and an **</html>** tag to indicate the end of the HTML page.

What happens when the designers decide that white-on-blue text is passe, and that they would rather have a more modern look (along the lines of *Wired* magazine) with orange text on a green background? It would not be very difficult for the designers to perform a global "search and replace" on the **<body>** tags appearing within the HTML files on the site. To modify each of the CGI programs on the server is much trickier.

A Simple Solution

One solution is to put all of our design-related variables in a library module that we can import into our programs. Here is an example of such a module called **SiteDesign**:

```
#!/usr/bin/perl -w
package SiteDesign;
$background_text = "white";
$foreground_text = "blue";
1;
```

The above module is named by the **package** statement. Following that statement, variables and functions are assumed to begin with the string **SiteDesign::**. To avoid problems with the package names when variables are imported into a program, we have turned off the normally helpful construct **use strict**.

Assume that the above code is placed in a file named **SiteDesign.pm**, and the file is placed in a directory named by the special Perl variable **@INC** (the list of directories in which Perl modules are located). Our programs should now be able to include this library with the statement:

```
use SiteDesign;
```

In other words, we could rewrite our "Hello, world" program as:

```
#!/usr/bin/perl -w
use strict;
use diagnostics;
use CGI; # Available from http://www.perl.com/CPAN
use SiteDesign;
# Create an instance of CGI
my $query = new CGI;
# Send an appropriate MIME header
print $query->header("text/html");
# Begin the HTML, with our colors indicated
print $query->start_html(
        -title => "Hello, world!",
        -bgcolor => $SiteDesign::background_text,
        -fgcolor => $SiteDesign::foreground_text);
# Send our message
print "<P>Hello, world!</P>\n";
# End the HTML
print $query->end_html;
```

This code is certainly an improvement over the first version of our program, in that the HTML produced by our programs can be changed without having to

modify the programs. Existing CGI programs do have to be changed so that they make use of SiteDesign.pm—but you only have to change your existing code once, rather than each time the site's design changes.

This approach is useful in many ways, but it does not solve all of the problems. While we have reduced the amount of work that a site's programmers need to perform each time the designers change their minds, we have not eliminated it entirely. The designers still have to come to the technical staff each time they wish to make such changes.

Furthermore, there is a practical limit to the number of ways in which we can affect our programs' output by setting variables. We could add a variable indicating which image, if any, should be displayed at the top of each page, another variable indicating whether an image should be displayed at the bottom of the page, another variable indicating the font size, yet another for whether the first paragraph should be centered, and so forth, ad infinitum. Sure, it would still be easier to change these variables than to change the output of each CGI program, but this solution does not scale well to a large number of variables. Would you want to be the programmer asked to modify 30 configuration variables each time the site's design was changed?

One possible solution to this problem is to put the variables in a configuration file, similar to the quiz file that we have discussed over the last few months. Such a file, particularly if it were masked by an interface consisting of CGI programs and HTML forms, would allow designers to modify the site's design without having to bother the programmers. However, designers would still have to deal with the large number of configuration variables, as well as understand what they mean. And programmers would still have to write code taking all sorts of styling possibilities into account.

In other words, the use of variables to indicate styling is better than nothing at all but is far from a perfect solution. What we would like is a way of creating pages of HTML that could be modified by designers, and also gives the possibility of executing code within those pages of HTML.

Templates

Luckily, we can create such hybrid Perl/HTML pages using the **Text::Template** Perl module, written by Mark-Jason Dominus. This module, available from CPAN at http://www.perl.com/CPAN/, allows us to take such hybrid files, evaluate the Perl parts, leave the pure HTML alone and send the results to the user's web browser. While the **template** module is identified as beta software and is not guaranteed to be stable, I have been using it for some time and have not encountered any problems. (I wish that I could say that about some of the commercial software that I use.) Although The **template** module is not designed

to work exclusively with HTML pages, it is in this area that I have found it to be highly useful.

Templates are pages of HTML that contain zero or more pieces of Perl code. (Thus, a plain HTML file is also a template, although such files don't do anything special.) The Perl code is contained inside the curly braces that Perl uses to identify blocks within programs. For example, here is one template that displays the time of day as recorded on the server:

```
<HTML>
<Head>
<Title>Welcome to our site</Title>
</Head>
<Body>
<P>Welcome to our site!The time is now
{localtime;}
</>P
</Body>
</HTML>
```

At first glance, the above template appears to be HTML and nothing more. If you look within the curly braces ({ }), you will see Perl code hiding there. In this particular case, we have used the Perl function “localtime”, which prints out the time and date using the standard Unix format.

Because the above file looks and acts like HTML—it **is** HTML, after all, except for the Perl code—we can give it to our designers, who can change the layout in any way they might like. If they wish to insert an image before/after the time or if they wish to center the time of day, they can do so by using the familiar HTML tags. The site's programmers merely have to stress the importance of not modifying the text contained within the curly braces, which should be off-limits to them. By the same token, the site's programmers should only modify the code contained within the curly braces, since that is the portion for which they are responsible.

By using templates, we get the best of both worlds. Pages can contain programs, and thus, can modify their output depending on circumstances, while styling is still determined by the HTML surrounding the blocks of code.

Writing templates is admittedly something that takes a bit of time to grasp; however, the principles of writing templates are easy to understand. As mentioned above, anything within curly braces is considered to be Perl code and is replaced by the results of its evaluation. Thus, the expression:

```
{ 2 + 2; }
```

returns 4, and the expression:

```
{
        $browser = $ENV{"HTTP_USER_AGENT"};
```

```
        $outputstring = "<P>You are using \"$browser\"
as your browser.</P>\n";
}
```

returns a string telling the user which browser he is using, bracketed by HTML “paragraph” tags.

It is also possible to make calculations in one block of Perl and to use the results of those calculations in a later block. Thus, we can create the following:

```
<HTML>
<Head>
<Title>Welcome to our site</Title>
</Head>
{
$time = localtime;
$browser = $ENV{"HTTP_USER_AGENT"};
}
<Body>
<P>Welcome to our site! The time is now
{ $time; }
</P>
<P>You are using {$browser;} to view our site.</P>
</Body>
</HTML>
```

In this code, we use the first block of Perl to assign variables needed in the rest of the template. It might seem a bit contrived but can be of great help when creating large, complicated templates to set up a number of variables in the first block and then to refer to them in subsequent blocks.

If we are not careful when writing blocks of code, we can accidentally insert some extraneous characters into our resulting page of HTML. In the above example, the first block of code assigns values to variables. The code block itself returns the value of **$browser**, since that was the last variable assignment. In other words, our users see the name of their browser twice—once where the first block sits and the second, where we might expect to see it, in the third Perl block.

In order to avoid such problems, I generally use a variable named **$outputstring**, which is used solely for the purpose of sending output to the resulting page of HTML. At the beginning of each block, I assign **$outputstring** the empty string (**""**), ensuring that it is not tainted by values from previous blocks. The last line of each block is then set to **$outputstring;**, which evaluates to the value of **$outputstring** and is sent to the user's browser. In between these two uses of **$outputstring** I can perform any calculations that I want—and anything that I want to send to the user is simply concatenated onto the current value of **$outputstring**.

Since CGI variables are actually environment variables and child processes inherit environment variables from their parents, we can also access CGI variables from within our templates. We have already seen this in the above

examples, when we retrieve **$ENV{"HTTP_USER_AGENT"}**, which should return the identifying string that web browsers send to web servers along with their document requests.

Because the code inside templates is full-blown Perl, we can use all of the techniques and code that we ordinarily use, including the use of library modules for databases, centralized libraries of code, and just about anything else available.

Of course, you need to be sure that your code is debugged before releasing it on an unsuspecting public. It is quite embarrassing to create a template and put it out in a public area of your web site, only to discover a bug that causes the entire template to crash. Actually, the template won't crash; the **template** module is smart enough to catch problems and point them out on the resulting page of HTML. Debugging templates can be tricky, so be sure to allocate additional testing and debugging time whenever you use templates rather than straight CGI programs.

The Template Wrapper

So, how do we turn a hybrid Perl/HTML template into plain HTML to be sent to the user's browser? If users were shown these templates without some sort of translation, they would appear as HTML files with the Perl reproduced verbatim on the user's screen. This display is obviously not desirable.

The key is to have a CGI program, called wrapper.pl, to take the name of a template in its query string (i.e., the argument that a CGI program receives following the question mark in the URL). Once it has received the template name, wrapper.pl creates an instance of **Text::Template** and instructs that module to perform the magic necessary to turn our template into a page of HTML.We can then send the resulting HTML to our user's browser. As far as the user is concerned, the page was and is HTML; he does not know that we have used a template to create our output.

Here is a simple version of wrapper.pl:

```
#!/usr/bin/perl -
use strict;
use diagnostics;
use CGI;
use Text::Template;
# Create an instance of CGI
my $query = new CGI;
# Send an appropriate MIME header
print $query-header("text/html");
# Get the name of the template
my $file = "/home/httpd/html/templates/" . $query-param("keywords");
# Create an instance of template
my $template = new Text::Template(-type => FILE,
        -source => $file);
# Perform the evaluation, and send the results
```

```
# to the user's browser
print $template-fill_in;
```

This program may appear quite simple, but we have hidden a great deal of depth within our calls to **Text::Template**—first when we open the file and when we ask the **Template** object to evaluate each of the small Perl programs inside the indicated template, it does so. Finally, we take the results of that evaluation and send them to the user's browser with the **print** statement.

Assuming the directory in which templates are stored not only makes the resulting URLs shorter but also makes your site somewhat more secure, since outsiders will not know your file system. It is also a good idea to remove any references to the parent directory (represented with "..") in the filename passed to wrapper.pl, so as to avoid turning our program into a convenient way of looking at all of the files on the server's hard disk. One easy way to do this is to replace the original assignment of **$file** with the following two lines:

```
# Get the name of the template
my $file = "/home/httpd/html/templates/" . $query->param("keywords");
# Remove possible security problems
$file =~ s|/\.\./|/|g;
```

This will remove attempts to ascend one or more directories, making it more difficult for someone to spy on the contents of our server.

Thus, if we have a template named /home/httpd/html/templates/test.tmpl and a site called www.oursite.com, we can view the template in translated form by using the URL http://www.oursite.com/cgi-bin/wrapper.pl?test.tmpl. If we have another template in the same directory named foo.html, we can view it using the URL http://www.oursite.com/cgi-bin/wrapper.pl?foo.html.

One odd note that you should remember when creating templates is the fact that they are effectively served out of the CGI directory on your server (usually called **cgi-bin**). In all of the above templates, this does not make any difference. If our templates were to incorporate images whose URLs were named relatively (i.e., without a leading slash) rather than absolutely (i.e., with a leading slash), this could cause a problem.

For example, it is quite common for HTML files to be placed in one directory, while the images used by those files are placed in a subdirectory, perhaps named "images". In order to create an HTML file with an image inside of it, we could do the following:

```
<HTML>
<Head>
<Title>Example of image</Title>
</Head>
<Body>
<P>This is a sample Web page, containing an image.</P>
<img src="images/graphic.gif">
```

```
</Body>
</HTML>
```

But if we were to take this same file and feed it through wrapper.pl, the image would no longer appear. That's because the “image” subdirectory exists relative to the directory in which the HTML file resides, rather than the directory in which the CGI program resides.

One quick solution to this problem is to use the **<base>** HTML tag, with a URL other than the one under which it was invoked. The **<base>** tag looks like:

```
<base href="http://www.oursite.com/text/english/">
```

With this tag in place, our browser will know to load the image in the above template from http://www.oursite.com/text/english/images, regardless of whether the document was loaded from within the CGI directory or the original HTML directory. The problem with this approach is that it makes it more difficult to move files and directories to other places on the site—a trade-off that is often worth making.

One word of warning before I conclude. Normally, access to the CGI directory and to the programs contained within it is restricted to a small set of programmers who can be trusted to write and modify code on your system. With templates, that group is suddenly expanded to include all of the site's designers, who could theoretically modify the code within a template to perform malicious acts. Remember that since templates include code, it is a good idea to restrict access to the directory containing the templates, rather than granting it to everyone on your system.

In short, templates are a useful way to separate the design of a web site from the CGI programs it contains. By using them wisely, you will give everyone more freedom to do what they enjoy, as well as what they do best.

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel, who has been using the Web since early 1993. In his spare time, he cooks, reads and volunteers with educational projects in his community. You can reach him at reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #40, August 1997

Readers sound off.

Using PPP

I have read many articles about PPP connections. The descriptions were huge and targeted complex tasks. That's why I always delayed configuring my system for a simple PPP connection.

That was true until I read the article “A 10-Minute Guide for Using PPP” in the April issue of *LJ*. I followed the instructions (just changed the init-string for my modem) and...it worked at once. Wow! Thanks very much to Terry Dawson for his excellent article and to *LJ* for publishing it. —Andreas Zisowsky, Berlin zisowsky@fokus.gmd.de

Exit Xenix, Linux is here.

In the March 1997 issue of *LJ*, E. Leibovitch wrote an article entitled “The Death of Xenix” about the opportunities offered by the near death of Xenix. Undoubtedly, *LJ* is in touch with its readers, or at least one of them.

I would like to tell you about my experience. A few years ago, the library of a high school (Lycee Victor Hugo, Colomiers, France) was computerized by means of an AT386PC with Xenix as OS and dedicated software. This system is now collapsing.

As a technical engineer working for the CRDP (Centre Regional de Documentation Pedagogique—a government organization acting in the area of educational services), I proposed that the existing system be replaced with an actual PC box with Linux as OS and the existing Wyse terminals should be reused. Our goal was to expand by installing an Intranet service to distribute the library databases into all the lycee (schools) using the existing Ethernet-based pedagogic network.

I encountered a few difficulties caused by:

1. lack of personal knowledge of the necessary setup,
2. inability to find not-too-new hardware (yesterday's software doesn't always work on today's hardware),
3. problems compiling the dedicated software: it worked first under IBCS, but I thought it would be better to get a genuine version. I am now in the test assembly stage.

Many thanks to *LJ* and its team for helping to sustain the Linux movement. —Jean Francois Bardou France devpoly@crdp231.crdp-toulouse.fr

PPP Control for Mortals

I tried your C program as shown in *LJ* in the article “Safely Running Programs as root” by Phil Hughes in the May 1997 issue. It compiled and worked like a champ. Keep it up; that sort of program is very helpful to strugglers like myself. Looks to me as though I could simply add it to the root menu in FVWM to bring it up and take it down. I'll let you know if it works. Thanks. —Jim Smith jim@oz.net

Bristol Zoo Location

On page 14 of May's *LJ* (*From the Editor*), you stated that Bristol Zoo is in Swansea. In fact, Bristol Zoo is in Bristol (75 miles from Swansea).

The confusion may have arisen because Alan Cox, at least at one point, had a Swansea University e-mail address and may still live in Swansea. I've never met Alan, so this is mere speculation. —Martin Radford M.P. Radford@exeter.ac.uk

Yes, the press release from Alan Cox originated from Swansea, and I assumed quite wrongly that the zoo was located there. Sorry —Ed.

Linux Appliance

I read your article “Linux—The Internet Appliance” in the April issue of *Linux Journal*. It is a subject I have also thought about and agree with your main points.

I think you missed one key point: the users you target probably don't want a floppy or a CD-ROM. A disposable small IDE hard disk would serve your machine much better. What do I mean by disposable? One that can be changed

out without disrupting their service. One that is automatically backed up by the ISP. One that serves the following needs:

- boot device
- swap space
- web-page cache
- off-line letter writing
- configuration information

In short, performance. Changing out the hard disk (to one that has been freshly dd(1)'d from a master, followed by a 10-second transfer of the account's setup from the ISP database) will be almost transparent to the customer, except that all the caches are empty, and therefore, they will slog along for a while like everybody else at home. When the caches fill up (200MB is cheap), all the usual pages and images will work as we rich, ethernetted folk expect.

Besides, by dropping the CD-ROM and floppy, and going to a smaller case and power supply, I think the cost total (including $200 monitor) is closer to $600 today than the $800 you suggest.

- 80 Fast 486 motherboard w/CPU
- 50 8M RAM (we have swap, remember?)
- 50 1M Cirrus video card (intended 800x600 16-bit mode)
- 120 540M IDE drive (buy out a warehouse of discontinued models)
- 60 28.8 modem
- 40 Sound card “Hi, my name is Leenus Torrrvalds and I pronounce Leeenux as Leeenux”
- 50 Case, Power supply, Keyboard, Mouse, lousy speakers
- 200 14 inch 800x600ni monitor
- 50 Assembly and testing
- 600 Total

Perilously close to the $500 target, right? In volume, you could probably get there now by contracting a custom motherboard, with embedded video, modem, sound, and power supply. It would then no longer be a PC-compatible (no ISA slots), but anyone who wanted that stuff could trade in for a “real” PC. The assumption of your article, which I support, is that there will be a large volume of people who don't want to fuss with it—just plug it in and use it. — Larry Doolittle ldoolitt@jlab.org

Bliss Misinformation

1. McAfee didn't find it [i.e., the Bliss virus, mentioned in *From the Editor*, May 1997—Ed.]—they were told about it. The author announced the fact that his Trojan had “accidentally” got out. Was it really an accident? Who knows?
2. It didn't “spread” to Linux systems. The released version of the Trojan was targeted specifically at Linux, although the author confirms OpenBSD, NT and other compile builds [with the Trojan] are trivial.

Bliss is a very simple Trojan. It sits on the front of files, copies itself into other stuff when run and spreads in that way. As such, not doing things as root will help a lot. Basic common sense like using PGP-signed packages and not installing random binaries as root also helps. —Alan Cox alan@cymru.net

Script Listings

I look forward to each issue of *Linux Journal* and usually read it from cover to cover the day it arrives in my mailbox. Your article on page 10 of the May issue, “Safely Running Programs as Root”, was very helpful. Before, my method of logging in to the Internet was to log in to my computer as **root**, start **ppp-go**, run **ifconfig** and edit /etc/hosts, then switch to another virtual terminal, log in with my **davidm** user name and fire up the X-server and Netscape.

Whenever I wanted to disconnect from my access provider, I would have to switch to **su** and run **ppp-off**. Now, things are much simpler. I grabbed your listing from the FTP site and in five minutes had your ppp.c program up and running. I am able to open and close my access connection at will without having to switch to root. Thanks for the story and for putting the code listings on your FTP server.

I thoroughly enjoy using Linux and reading *Linux Journal*. If you get a chance, check out my column on Linux at the following URL: http://www.charleston.net/entertain/click3.html. —David W. MacDougall, South Carolina davidm@Charleston.Net

Linux in Schools

In my free time, I read a whole slew of computer magazines on subjects ranging from Windows NT to LANs. One thing that struck me the other day was how much fun it was to read the *Linux Journal*. It seems that every columnist writes with such enthusiasm for the subject. This is a refreshing change from the other mainstream magazines, which seem to complain about *everything*. Your authors enjoy writing about how far Linux can be pushed and how it can be reshaped into something new.

Granted, Linux is not a high-dollar commercial OS like the others, but I believe that is to its advantage. You have to be amazed with the way it was, and is, being developed. It shows how a large diversity of people can come together for a common cause (one that didn't include money) to create an extremely fun and useful product.

I have specified Linux as the OS for one of the servers (Pentium Pro 200 MHz, 128MB RAM, 4GB HD) in the public school system I work for. I look forward to using it both at home and work. I'd like to hear from other EdTech folks who use Linux in a school environment. —Rob Bellville, Millbury, MA rob@millbury.k12.ma.us

Linux—Not for Newbees?

I am still a Newbee after 18 months of working with Linux off and on. I feel I must comment on *Stop the Presses* by Phil Hughes in your April issue: "Usenix/Uselinux in Anaheim" on page 8. In that report it was stated that Linus Torvalds hinted at "world domination" with Linux.

After my venture into the Linux OS, it seems to me that there is still a long way to go. It appears that Linux has been authored by a large number of academics who each make a mark on the system. I have found a large number of help files to be out of sync with the code they are trying to explain. I have just started working with **pppd** after signing up with an ISP provider and find the various configurations expressed in the files confusing. It is true I am not a genius, but if one wants to have a system appeal to the "regular joes" out in the real world, setting the system up will have to be made easier.

I am certain I will eventually sort out my problem. It will take a lot of work and learning on my part, which I don't mind—I enjoy sorting out a difficulty and getting it solved. I am going to stick it out until I can connect with my Linux box and get some work done out on the Internet. To that end, I have Linux on separate hard drives on two machines so that I have a backup in case I corrupt one.

Anyhow, keep up the great work in *LJ*, as I do find it very helpful to keep abreast of what is going on out there. —Kurt Savegnago ksaves@prairienet.org

Article Choices

I used to like *Linux Journal* because of the low-level, programmer-oriented topics in it. Unfortunately, the past few issues have been on the boring parts of Linux, such as platforms and networking. The majority of Linux users are not into the many platforms or top-of-the-line systems, but into the hacking of the kernel and the fun of how it works. I am more than willing to have some articles

included on the higher levels of technology, but I would also like to see many articles on the hacking of the kernel and nifty applications for Linux.

Thank you for putting out such a good magazine. I would hate to see it turn into just another *PC Magazine* type of magazine. —Patrick Temple
patemple@erols.com

Platforms and networking certainly seem to be popular subjects, judging from the response we get to articles about them. Although we sometimes miss, we do always try to have a Kernel Korner for the kernel hackers as well as articles on new applications. We do our best to present a balance of topics —Ed.

Archive Index Issue Table of Contents

Advanced search

From the Editor

Marjorie Richardson

Issue #40, August 1997

On the formation of GLUE.

In the *Linux Journal 1997 Buyer's Guide*, SSC announced the formation of GLUE—Groups of Linux Users Everywhere. GLUE was implemented to provide a world-wide member group for Linux User Groups. Since three months have passed, I felt an update would be in order. Complete information about the advantages to GLUE membership can be found at its web site, http://www.ssc.com/glue/.

Lydia Kinata, SSC Product Specialist, tells me there has been a good response to GLUE from both user groups and vendors. At the time I talked to her (May 1), SSC was in the process of setting up an e-mail list server for use by the groups. Lydia was particularly excited about the decision by Enhanced Software Technologies to provide a free copy of the BRU 200 Backup and Restore Utility to each GLUE member group, as well as a 10% discount on the BRU software to individual members of the user groups. Enhanced Software (http://www.estinc.com/) is located in Tempe, Arizona and recently became a corporate member of Linux International.

London Times

As I'm sure everyone has heard by now, in April *The London Times* printed an article by David Hewson 1 which trashed Linux and the “geeks” who use it. I just reread that article to determine if I needed to respond to it in some way. Frankly, after reading through phrases like “nasty piece of digital scurf”, “that old computer donkey known as Unix” and “a certain breed of bug-eyed computer user”, I was laughing too much to take it seriously. However, I did find the notion of “Bill Gates quivering in his boots at the idea that Linux will ... kick Microsoft Windows off the everyday desktop” rather appealing. At any rate I found Hewson's rantings amusing and, most certainly, nothing to incite a flame war. (*The Sunday Times* - 20 April 1997, *Sounding Off: Linux, the PC Program*

from Hell, by David Hewson, http://www.the-times.cp.uk/news/pages/resources/libraryl.n.html?1032133.)

New Printer

From January of 1996 through June of 1997 *Linux Journal* has been printed by R. R. Donnelly in Senatobia, Mississippi. Beginning with last month's issue, it is now printed by Century Publishing in Post Falls, Idaho. One reason we made this change was to have a printer in the same time zone. More importantly, we wanted a printer closer to us so that we could reduce the lead time in producing the magazine. Century fit both these needs. Century is also known to us: Century was the first printer for *Linux Journal*, and SSC has continued to use them for printing the SSC catalog.

The new subscriber will be the main beneficiary of the shorter lead time—he'll get his first issue sooner. Other benefits will include later deadlines for articles and advertising materials.

We believe this change to be a positive move on our part and look forward to a long and harmonious working relationship with Century.

Coming Up

Our next couple of issues will be focusing on *Education and Training Using Linux* and *Linux as a Development Platform*. We'll round out the year with another *Graphics and Multimedia* issue, then focus on *System Administration* in December. We have a lot of good articles being written for the first three, but commitments for system administration articles are lagging. So, I'd like to remind all you authors that we keep a “wish list” of articles on our web site, http://www.ssc.com/lj/wanted.html. Check it out, and if you find one you're interested in, write us at info@linuxjournal.com.

For some time now, we have wanted to run a cartoon each month. Although we have someone who can draw cartoons, he knows nothing about Linux. Send your favorite one-line Linux jokes to info@linuxjournal.com, and if we can, we'll turn them into cartoons.

Archive Index Issue Table of Contents

Advanced search

Linux Trademark Dispute

Phil Hughes

Issue #40, August 1997

Being on the Board of Directors of Linux International, I was a participant as well as an observer in the battle.

At this point it looks like the good guys will win. As I write this (June 4) there is a verbal agreement which will give the Linux community clear title to the Linux name. By the time you read this article there should be a final, written agreement. For details, select the Hot Linux News button on our web page at http://www.ssc.com/lj/.

For those of you new to the trademark dispute, let me fill in a little history. In 1996, Linux vendors (including us) started receiving letters from an attorney representing Mr. William Della Croce, Jr. requesting royalties for our use of his trademark. Investigation revealed that he did in fact have a trademark on the Linux name.

The Linux vendor community decided to fight the battle together and, through Linux International, enlisted the services of G. Gervaise Davis III of Davis & Schroder (http://www.iplaywers.com). Gerry Davis had already jumped on the bandwagon because he knew of the Linux effort. His willingness to take the case because he cared cut our costs substantially.

Being on the Board of Directors of Linux International, I was a participant as well as an observer in the battle. While not everyone agreed with every decision along the way, we all shared the common goal of getting the Linux trademark clearly into the hands of the Linux community. Even with our diverse backgrounds we managed to pull together the necessary information and resources to present a common front.

For example, when we found out that Mr. Della Croce's trademark was filed in 1994, Adam Richter of Yggdrasil Computing jumped forward with information that he was shipping Linux on CDs in December, 1992. Also, a few hundred

thousand copies of *Linux Journal* published in 1993 certainly had to help our case.

Where Are We Now?

First and foremost, we have proved that while we may have competing commercial interests, we can work together for the common good of the Linux community. I also think that while this action was costly to Linux vendors it has helped Linux become legitimate in the eyes of the non-believers. Besides the vendors who are members of Linux International that supported the effort, we have received letters from many of our readers asking if there was a place to send money to help support the effort. This shows the sense of cooperation that made Linux possible in the first place.

Thanks to everyone that helped.

Archive Index Issue Table of Contents

Advanced search

New Products

LJ Staff

Issue #40, August 1997

Cyclades-Z Multiport Serial Cards: S.u.S.E. Linux Version 4.4.1, LinkScan Version 2.1, AllConnect, 3DGO, and more...

Cyclades-Z Multiport Serial Cards

Cyclades Corporation announced a new family of multiport serial cards for the ISP and corporate remote access needs. The Cyclades-Z family sustains effective serial speeds of up to 460Kbps and is available in low-cost boards with 8 ports to rack-mountable models expandable to 64ports/slot. The product is being introduced with support for both Linux and NT operating systems. The 8-port model is priced at $623, but ISPs and first-time buyers can get the board at the promotional prices ($399 for an 8-port model) for a limited time.

Contact: Cyclades Corporation, 41934 Christy Street, Fremont, CA 94538, Phone: 510-770-9727 ext. 206, Fax: 510-770-0355, E-mail: info@cyclades.com, URL: http://www.cyclades.com/.

S.u.S.E. Linux Version 4.4.1

S.u.S.E. announced S.u.S.E. Linux Version 4.4.1, a distribution of the Linux operating system, which brings 32-bit multitasking Unix to Intel-based PCs. The new version contains X Window support via XFree86 3.2A, RPM packages, modular kernel 2.0.29, Teles 16.3 ISDN card support and a DOS emulator that is loadable without recompiling the kernel. Software highlights include LNX, the Java Development Kit and an HTML-based help system. The distribution features three CD-ROMs, a 400-page reference book and the YaST setup and management tool.

Contact: S.u.S.E., LLC, 458 Santa Clara Avenue, Oakland, CA, 94610, Phone: 510-835-7873, Fax: 510-835-7875, E-mail: jgray@suse.com, URL: http://www.suse.com/.

LinkScan Version 2.1

Electronic Software Publishing Corporation (Elsop) introduced new features in LinkScan version 2.1 that enhance its power and flexibility for checking Internet and Intranet web sites. Elsop has added a number of features, including support for server aliases, server redirections and selective execution of CGI scripts. LinkScan also supports proxy servers. Elsop's LinkScan reports and site maps may be viewed using any standard Web browser. Free fully functional evaluation copies of LinkScan can be downloaded from the company's web site at: http://www.elsop.com/.

Contact: Electronic Software Publishing Corporation, 1504 #8-00200 Main Street, Gardenerville, NV 89410-5273, E-mail: linkscan@elsop.com, URL: http://www.elsop.com/.

AllConnect

Empress Software, Inc. announced support for Digital Equipment Corporation's AllConnect for Unix, a program designed to assist software developers in the integration and migration of Unix and Windows NT applications. Empress has also recently unveiled a combination of RDBMS development tools, including the Embedded Empress RDBMS Developer's Toolkits for Unix and NT. The toolkits include the Empress RDBMS, the Empress HTML Toolkit for interfacing database driven applications to the Internet/Intranet and the Empress ODBC Server which enables use of the RDBMS with 3rd-party ODBC-compliant tools. The Toolkits are available for $1,000 for 2 users on NT, $5,600 for typical workstations and $22,400 for mid-range servers.

Contact: Empress Software Incorporated, 6401 Golden Triangle Drive, Greenbelt, MD, 20770, Phone: 301-220-1919, Fax: 301-220-1997, E-mail: sales@empress.com, URL: http://www.empress.com/.

3DGO

ELECTROGIG, a software company specializing in ray tracing, animating and modeling, announced that 3DGO (version 3.2) is available for the Linux PC platform and a beta version is available for the MkLinux platform. (3DGO was developed on the SGI Unix platform.) A demo-version of 3DGO for Linux can be downloaded from ftp://ftp.gig.nl/demo/.

Contact: ELECTROGIG Technology, Phone: 31-20-4624600, Fax: 31-20-4650990, E-mail: info@gig.nl, URL: http://www.gig.nl/products/prodinfo.html/.

XRT Products and Metro Link Motif

KL Group, Inc. and Metro Link, Inc. announced today that Metro Link's Motif for Linux will now include KL Group's XRT Professional Developer's Suite free. XRT PDS is a recently announced suite of market-leading widgets and utilities for professional Motif developers. Metro Link Motif is popular with Linux users who wish to implement the industry standard Motif Graphical User Interface. Metro Link Motif 2.0.1, including the free XRT PDS has a list price of $199. The free version is functionally identical to KL Group's standard product, except that it does not come with printed documentation. Complete on-line documentation is included.

Contact: Metro Link, Incorporated, 4711 North Powerline Road, Fort Lauderdale, FL, 33309, Phone: 954-938-0283, Fax: 954-938-1982, E-mail: sales@metrolink.com, URL: http://www.metrolink.com/.

Contact: KL Group, 260 King Street East, Floor Three, Toronto, ON, Canada, Phone: 416-594-1026, E-mail:info@klg.com, URL: http://www.klg.com/.

MPAS

Strategic Forecasting, L.L.C. announced the release of MPAS (Multi-Platform Access System). MPAS provides resource management functionality similar to NT and Novell but at a lower cost. It runs on DOS, Windows 3.x and Windows95 clients and supports links to virtually any server platform. MPAS' single step login provides a cross-platform system interface allowing Windows and DOS users controlled, secure access to personal or shared files from any server on the network. The system administrator can re-map a user's virtual drives in a matter of minutes. Maintenance and system changes can be performed

without interference to the user's activities. MPAS version 1.01 is available at a special introductory price of $10 per client with a minimum 20 seat purchase.

Contact: Strategic Forecasting, L.L.C., 617 North Blvd., Suite 400 Baton Rouge, LA 70802, Phone: 504-267-5507. Fax: 504-267-4108, E-mail: info@mpas.com, URL: http://www.mpas.com/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

SATAN: Analyzing Your Network

Rob Havelt

Issue #40, August 1997

SATAN checks out your network for possible openings that hackers can use to invade your system.

The creators of SATAN (System Administrator's Tool for Analyzing Networks) feel that the reason most systems are vulnerable to attack is that most System Administrators don't think like system crackers. Actually, thinking this way may require you to look at simple, seemingly harmless network services in a new light.

To the system cracker, basic network services are a doorway into a computer system. So, before you go into the office one morning and see a bunch of unusual **utmp** entries, it is beneficial to check just how accessible those doorways are. A great starting point for checking the doors is to run a SATAN scan on your network.

Requirements

SATAN can be run on Linux with a few modifications. Requirements for running SATAN are:

1. A machine that can handle it—which means a box with a relatively fast processor (e.g., Alpha, 486) and at least 32MB of RAM
2. A recent distribution of the SATAN source code (satan-1.1.1)
3. Perl 5 or greater
4. A set of BSD-4.4 compatible include files, available from ftp://ftp.wooddimensions.com/webserv/security/linux/
5. A patch to fix the mistaken assumptions about how **select()** works in tcp_scan.c (tcp_scan.c.diff)
6. A WWW browser (A graphical browser like Netscape or Mosaic is preferable, but you can also use a text browser like Lynx.)

7. A C compiler like **gcc**

Building SATAN

When you have all of these elements, you can begin to build SATAN. First you want to untar the archive. Just issue the command:

```
zcat satan-1.1.1.tar.Z | tar xvf -
```

This command creates a subdirectory called satan-1.1.1. Next, apply the patch to tcp_scan.c. I tend to use Emacs for patching, as it makes everything a little more visual. Load the patch into one buffer, tcp_scan.c into another, and choose **Patch**. Of course, using the patch command works fine too. Now, untar the BSD-4.4 compatible include files. The easiest way to do this is to **gunzip** the archive and move it into the root of the satan-1.1.1 directory, and then type:

```
tar xvf BSD-4.4.includes.tar
```

The archive will expand into the include/netinet/ directory.

Having done this, you are ready to compile SATAN. The program comes with a script, called **reconfig**, which will configure it on your system. Any Linux user who has used SATAN before knows that bash has trouble with the syntax of this script. The easiest way around this is to type:

```
perl reconfig
```

at the command prompt, rather than:

```
./reconfig
```

The **reconfig** script will detect your web browser and Perl and compile the SATAN binaries. If it detects the wrong web browser, edit the script config/paths.pl and change the line:

```
$MOSAIC="program name"
```

You are now ready to run SATAN. Type **./satan** at the command prompt, and SATAN will fire up your web browser.

If Netscape is your browser of choice, make sure you have a mime type defined for application/x-perl, and no suffix is defined for this type. Defining the suffix as .pl will result in errors every time you try to execute a script.

Stay Away from Precompiled Binaries

Even if all of the above steps sound like a major pain, you should still get the sources and build SATAN yourself. I would strongly urge you not to request,

post or use precompiled binaries of SATAN. SATAN must be run as root, and a bad or malicious build can do volumes of damage. There have already been several reports of Trojans found in builds for Linux. Building SATAN for Linux might take a few extra steps, but it is definitely worth the effort.

After the Scan

SATAN will dutifully scan your network and report back all the potential weaknesses that it finds—that is its job. It will even tell you how those weaknesses might be exploitable. It will not fix any problems or keep unwanted guests out—that is your job. No program can be a substitute for an astute Security Administrator.

To run a tight ship you must keep the crew in line, which means educating your users on the importance of a good password. (It's up to you whether you send out security memos, post in the MOTD or actually periodically attempt to crack /etc/passwd and lock out accounts you were able to crack.) Along with password education, educate your users on the dangers of keeping large .rhosts files in their home directories. The more unknown systems trusted, the greater the risk to your own system.

Finally, take a look at your system in the same way an educated cracker might. Subscribe to *2600* and *Phrack*, if your hacking skills are not up to snuff. Take a look at the network services you are running and think of possible ways you could exploit them. Read the latest CERT (http://www.cert.org/). advisories for all systems (as many common programs come from the same roots, they sometimes share the same weaknesses) and, using this information, periodically try to break into your own system. If you are new to system security or if you are unsure how to go about exploiting network services, try all the cookbook approaches used in such texts as *The System Administrator's Guide to Cracking* (included with the SATAN distribution). There are also a lot of IRC channels and web sites where hacking and cracking are discussed. Visit these sites and listen in or ask questions. Your users are depending on you to have the system up and running—with a little work, you won't disappoint them.

Rob Havelt is the Webmaster/Security Administrator at Wood Dimensions Ltd. in Detroit, MI, a full-service Internet and Computer Aided Design Firm (http://www.wdl.net/). He can be reached at rob@wdl.net. He is a self-proclaimed technology addict and can usually be found designing 3D graphics and virtual worlds.

Archive Index Issue Table of Contents

Advanced search

A Non-Technical Look inside the EXT2 File System

Randy Appleton

Issue #40, August 1997

How the EXT2 file system is organized on the disk and how it gets its speed.

Everyone wants a fast computer; however, not everyone realizes that one of the most important factors of computer performance is the speed of the file system. Regardless of how fast your CPU is, if the file system is slow, then the whole computer will seem slow. Many people who have very fast Pentium Pros with slow disk drives and even slower networked file systems rediscover this fact daily.

Linux has a very fast file system called the **Extended File System Version 2 (EXT2)**. The EXT2 file system was created by Remy Card (card@masi.ibp.fr).

Disk Layout

There are several objectives when deciding how to lay out data on a disk.

First and foremost, the data structure should be **recoverable**. If there is an error while writing data to the disk (like a user pulling the power cord), the entire file system should not be lost. Although losing the data currently being written is sometimes acceptable, losing all the data on the disk is not.

Second, the data structure must allow for an **efficient implementation** of all needed operations. The hardest operation to implement is normally the hard link. When using a hard link, there is more than one directory entry (i.e., file name) that points to the same file data. Accessing the data by any of the valid file names should produce the same data.

Another hard operation involves deleting an open file. If an application has a file open for access at the same time that a user deletes the file, the application should still be able to access the file's data. The data should not be cleared off the disk until the last application closes the file. This sort of behavior is quite

unlike DOS/Windows, where deleting a file results in immediate loss of access to that file by any application in the process of reading/writing to it. Applications exhibiting this type of Unix behavior are more common than one might think, and changing it would cause many applications to break.

Third, a disk layout should minimize seek times by **clustering** data on the disk. A drive needs more time to read two pieces of data that are widely separated on the disk than the same sized pieces located close to each other. A good disk layout can minimize disk seek time (and maximize performance) by clustering related data close together. For example, parts of the same file should be close together on disk and, also, near the directory containing the file's name.

Finally, the disk layout should **conserve disk space**. Conserving disk space was more important in the past, when hard drives were small and expensive. These days, conserving disk space is not so important; however, one should not waste disk space.

Partitions

Partitions are the first level of disk layout. Each disk must have one or more partitions. The operating system pretends each partition is a separate logical disk, even though they may share the same physical disk. The most common use of partitioning is to place more than one file system on the same physical disk, each in its own partition. Each partition has its own device file in the **/dev** directory (e.g., **/dev/hda1, /dev/hda2**, etc.). Every EXT2 file system occupies one partition, and completely fills it.

Groups

The EXT2 file system is divided into **groups**, which are sections of a partition. The division into groups is done at the time the file system is formatted and cannot change without reformatting. Each group contains related data. A group is the unit of clustering in the EXT2 file system. Each group contains a **superblock**, a **group descriptor**, a **block bitmap**, an **inode bitmap**, an **inode table** and finally **data blocks**, all in that order.

Superblock

Some information about a file system belongs to the file system as a whole and not to any particular file or group. This information is stored in the **superblock**, and includes the total number of blocks within the file system, the time it was last checked for errors and so on.

The first superblock is the most important one, since it is the first one read when the file system is mounted. The information in the superblock is so

important that the file system cannot be mounted without it. If a disk error occurred while updating the superblock, the entire file system would be ruined; therefore, a copy of the superblock is kept in each group. If the first superblock becomes corrupted, the redundant copies can be used to fix the error by using the command **e2fsck**.

Group Descriptors and Bitmaps

The next block of each group is the **group descriptor**. The group descriptor stores information on each group. Within each group descriptor is a pointer to the table of inodes (more on inodes in a moment) and **allocation bitmaps** for inodes and data blocks.

An allocation bitmap is simply a list of bits describing which blocks or inodes are in use. For example, data block number 123 is in use if bit number 123 in the data bitmap is set. Using the data and inode bitmaps, the file system can determine which blocks and inodes are in current use and which are available for future use.

Inodes and Such

Each file on disk is associated with exactly one **inode**. The inode stores important information about the file including the create and modify times, the permissions on the file and the owner of the file. The inode also contains the type of the file (regular file, directory, device file like /**dev/ttyS1**, etc.) and the location of the file on disk.

The data in the file is not stored in the inode itself. Instead, the inode points to the location of the data on disk. There are fifteen pointers to data blocks within each inode. However, this does not mean that a file can only be fifteen blocks long. Instead, a file can be millions of blocks long, thanks to the indirect way that data pointers point to data.

The first thirteen pointers point directly to blocks containing file data. If the file is thirteen or fewer blocks long, then the file's data is pointed to directly by pointers within each inode and can be accessed quickly. The fourteenth pointer is called the indirect pointer and points to a block of pointers, each one of which points to data on the disk. The fifteenth pointer is called the doubly indirect pointer and points at a block containing many pointers to blocks each of which points at data on the disk. The picture shown in Figure 1 should make things clear.

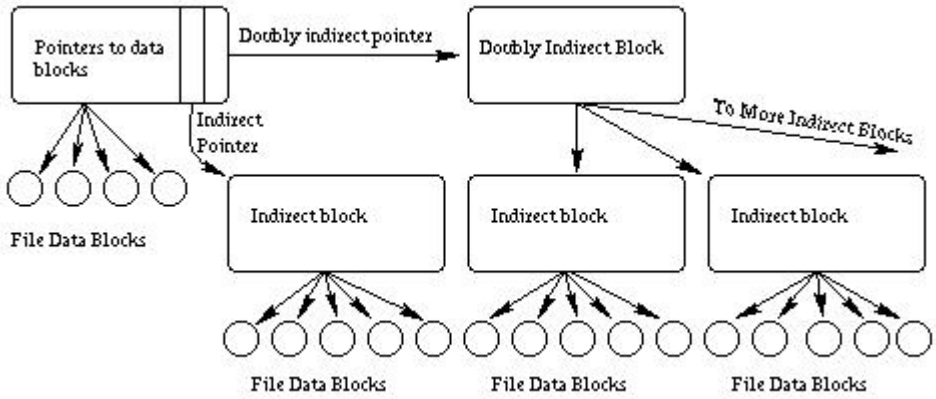


Figure 1. The pointers between an inode and its associated data.

This scheme allows direct access to all the data of small files (files less than fourteen blocks long) and still allows for very large files with only a few extra accesses. As Table 1 shows, almost all files are actually quite small; therefore, almost all files can be accessed quickly using this scheme.

Table 1. Occurrence of Various File Sizes

Inodes are stored in the inode table, which is at a location pointed to by the group descriptor within each group. The location and size of the inode table is set at format time and cannot be changed without reformatting. This means that the maximum number of files in the file system is also fixed at format time. However, each time you format the file system you can set the maximum number of inodes with the **-i** option to **mke2fs**.

Directories

No one would like a file system where files were accessed by inode number. Instead, people want to give textual names to files. Directories associate these textual names with the inode numbers used internally by the file system. Most people don't realize that directories are just files where the data is in a special directory format. In fact, on some older Unix systems, you could run editors on the directories, just to see what they looked like internally (imagine running **vi /tmp**).

Each directory is a list of directory entries. Each directory entry associates one file name with one inode number and consists of the inode number, the length of the file name and the actual text of the file name.

The root directory is always stored in inode number two, so that the file system code can find it at mount time. Subdirectories are implemented by storing the name of the subdirectory in the name field and the inode number of the subdirectory in the inode field. Hard links are implemented by storing the same

inode number with more than one file name. Accessing the file by either name results in the same inode number, and therefore, the same data.

The special directories "." and ".." are implemented by storing the names "." and ".." in the directory and the inode number of the current and parent directories in the inode field. The only special treatment these two entries receive is that they are automatically created when any new directory is made, and that they cannot be deleted.

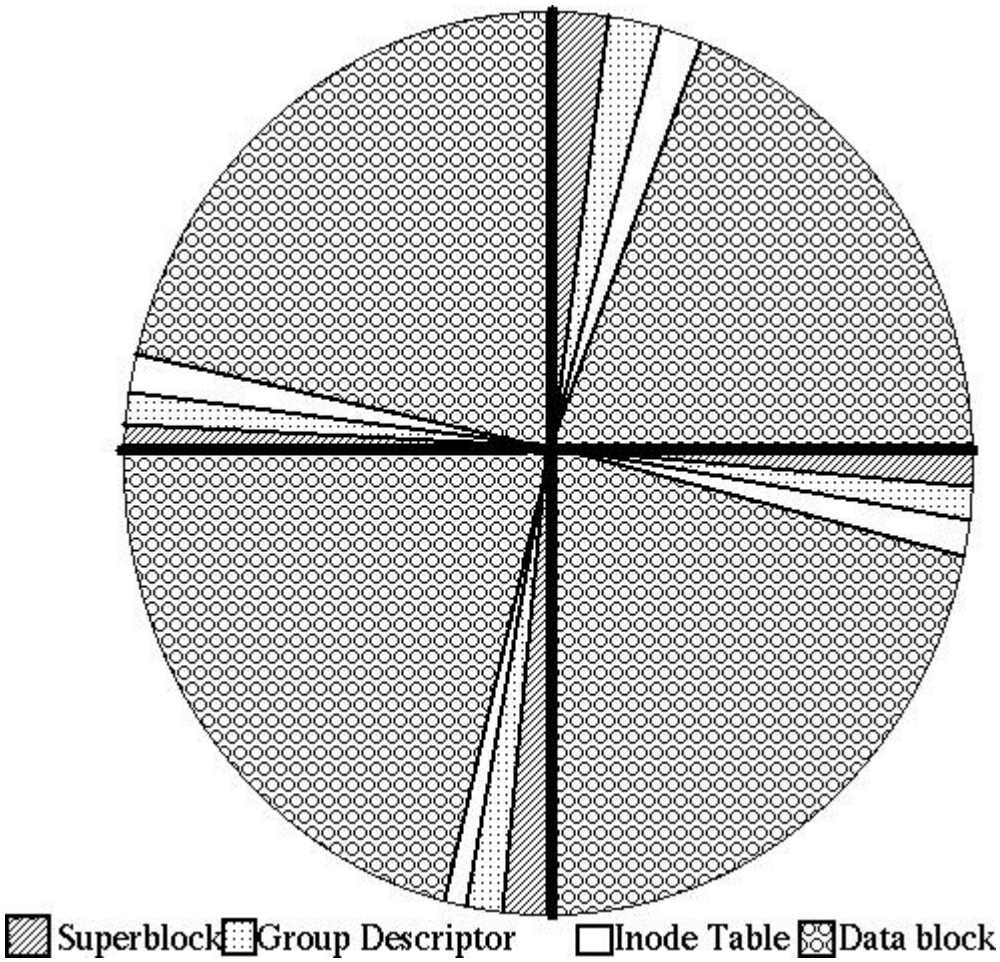


Figure 2. Layout of a disk with one partition and four groups.

The File System in Action

The easiest way to understand the EXT2 file system is to watch it in action.

Accessing a file

To explain the EXT2 file system in action, we will need two things: a variable named DIR that holds directories, and a path name to look up. Some path names have many components (e.g., **/usr/X11/bin/Xrefresh**) and others do not (e.g., **/vmlinuz**).

Assume that a process wants to open a file. Each process is associated with a current working directory. All file names that do not start with “/” are resolved relative to this current working directory and DIR begins with the current working directory. File names that start with “/” are resolved relative to the root directory (see **chroot** for the one exception), and DIR begins with the root directory.

Each directory name in the path to be resolved is looked up in DIR in turn. This lookup yields the inode number of the subdirectory we're interested in.

Next, the inode of the subdirectory is accessed. The permissions are checked, and if you have access permissions, this new directory becomes DIR. Each subdirectory in the path is treated in this fashion, until only the last component of the path remains.

When the last component of the pathname is reached, the variable DIR contains the directory which contains the file name we've been searching for. Looking in DIR we find the inode number of the file. Accessing this final inode tells us the location of the data. After checking permissions, you can access the data.

How many disk accesses were needed to access the data you wanted? A reasonable maximum is two per subdirectory (one to look up the name, the other to find the inode) and two more for the actual file name. This effort is expended only at file open time. After a file has been opened, subsequent accesses can use the inode's data without looking it up again. Further, **caching** eliminates many of the accesses needed to look up a file (more later).

Listing 1

When a new file or directory is created, the EXT2 file system must decide where to store the data. If the disk is mostly empty, data can be stored almost anywhere. However, performance is maximized if the data are clustered with other related data to minimize seek times.

The EXT2 file system attempts to allocate each new directory in the group containing its parent directory, on the theory that accesses to parent and children directories will be closely related. The EXT2 file system also attempts to place files in the same group as their directory entries, because directory accesses often lead to file accesses. However, if the group is full, the new file or new directory is placed in some other non-full group.

The data blocks needed to store directories and files can be found by looking in the data allocation bitmap. Any needed space in the inode table can be found by looking in the inode allocation bitmap.

Caching

Like most file systems, the EXT2 system relies very heavily on caching. A **cache** is a part of RAM dedicated to holding file system data. The cache holds directory information inode information and actual file contents. Whenever an application (like a text editor or a compiler) tries to look up a file name or requests file data, the EXT2 system first checks the cache. If the answer can be found in the cache, the request can be answered very quickly indeed without using the disk.

The cache is filled with data from prior requests. If you request data that you have never requested before, the data must first be retrieved from disk. Most of the time most people ask for data they have used before. These repeat requests are answered quickly from the cache, saving the disk drive much effort while providing the user quick access.

Of course, each computer has a limited amount of RAM available. Most of that RAM is used for other things like running applications, leaving perhaps 10% to 30% of total RAM available for the cache. When the cache becomes full, the oldest unused data (least recently used data) is thrown out. Only recently used data remains in the cache.

Since larger caches can hold more data, they can also satisfy a larger number of requests. The figure below shows a typical curve of the total cache size versus the percent of all requests that can be satisfied from the cache. As you can see in Figure 3, using more RAM for caching increases the number of requests answered from the cache, and therefore increases the apparent speed of the file system.

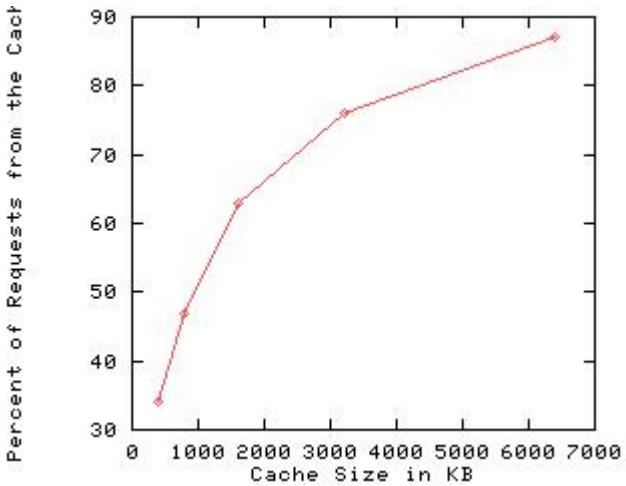


Figure 3. A typical curve of total cache size versus the number of requests satisfied from the cache.

Conclusion

As has been said before, one should make things as simple as possible, but no simpler. The EXT2 file system is rather more complex than most people realize, but this complexity results in both the full set of Unix operations working correctly, and good performance. The code is robust and well tested and serves the Linux community well. We all owe a debt of thanks to M. Card.

Sources for More Information

Randy Appleton is a professor of Computer Science at Northern Michigan University. He got his Ph.D. at the University of Kentucky. He has been involved with Linux since before version 0.9. Current research includes high performance pre-fetching file systems, with a coming port to the 2.X version of Linux. Other interests include airplanes, especially home-built ones. He can be reached via e-mail at randy@euclid.acs.nmu.edu.

Archive Index Issue Table of Contents

Advanced search

Big Brother Network Monitoring System

Paul M. Sittler

Issue #40, August 1997

Installing and hacking Big Brother, a web-based Unix network monitoring and notification system.

Figure 1. Big Brother (Sean MacGuire) is Watching

I wasn't bored: I don't have time to be bored. Texas Agricultural Extension Service operates a fairly large enterprise-wide network that stretches across hell's half acre, otherwise known as Texas. We have around 3,000 users in 249 counties and 12 district offices who expect to get their e-mail and files across our Wide Area Network. Some users actually expect the network to work most of the time. We use Ethernet networking with Novell servers at some 35 locations, about 15 have routers that are connected via a mixture of 56Kb circuits, fractional T1, frame-relay and radio links. We are not currently using barbed wire fences for our network, no matter what you may have heard.

I am privileged to be part of the team that set up and maintains the network. We do not live in a perfect network world—things happen. Scarcely a day goes by that we do not have one or more WAN link outages, usually of short duration. We sometimes have our hands full just keeping all the pieces connected. Did I mention that the users expect the mail and other software to actually work?

Cruising the USENET newsgroups, I read a posting about “Big Brother, a solution to the problem of Unix Systems Monitoring” written by Sean MacGuire of Montréal, Canada. I was intrigued to notice that Big Brother was a collection of shell scripts and simple C programs designed to monitor a bunch of Unix machines on a network. So what if most of our mission critical servers were Novell-based? Who cares if some of our web servers run on Macintosh, OS/2, Windows 95 or NT? We use both Linux and various flavors of Unix in a surprisingly large number of places.

System administrators often reported difficult installations and software incompatibilities with the monitoring software; thus, frustrated users often gave us our first hint that all was not well. We had cooked up a number of homemade monitoring systems; pinging and tracerouting to all the servers can be very informative. We even looked at a bunch of proprietary (and expensive) network monitoring systems. It is amazing how much money these systems can cost.

According to the blurb by Sean MacGuire on Big Brother:

> Big Brother is a loosely-coupled distributed set of tools for monitoring and displaying the current status of an entire Unix network and notifying the system administrator should need be. It came about as the result of automating the day to day tasks encountered while actively administering Unix systems.

The USENET news article provided a URL to the home site of Big Brother, http://www.iti.qc.ca/iti/users/sean/bb-dnld/. I pointed my browser to it and was rewarded with a blue image of a sinister face peering out under the caption “big brother is watching ”against a purple background. After my initial shock, I learned that Big Brother featured:

- Web-based status display
- Configurable warning and panic levels
- Notification via pager or e-mail
- Free and included source code

I was fascinated, especially by the last item: “Free and includes source code.” (I often tell people that Linux isn't free, but priceless.) So what could a priceless package do for me? What does Big Brother check?

- Connectivity via ping
- HTTP servers up and running
- Disk space usage
- Uptime and CPU usage

- Essential processes still running
- System-generated messages and warnings

Overall, very sensible. Looking for some “gotchas”, I found I would need a Unix-based machine, a functioning web server and browser (for the display), a compiler, Kermit and a modem line (for the pager). A web server was no problem, as we run many. A C compiler came with Linux, and we use Kermit on many machines with modems. So far, so good.

The Big Brother web site provided links to a few demonstration sites, and a link to download the program as well. I connected to a demonstration site and was greeted with an amazing display:

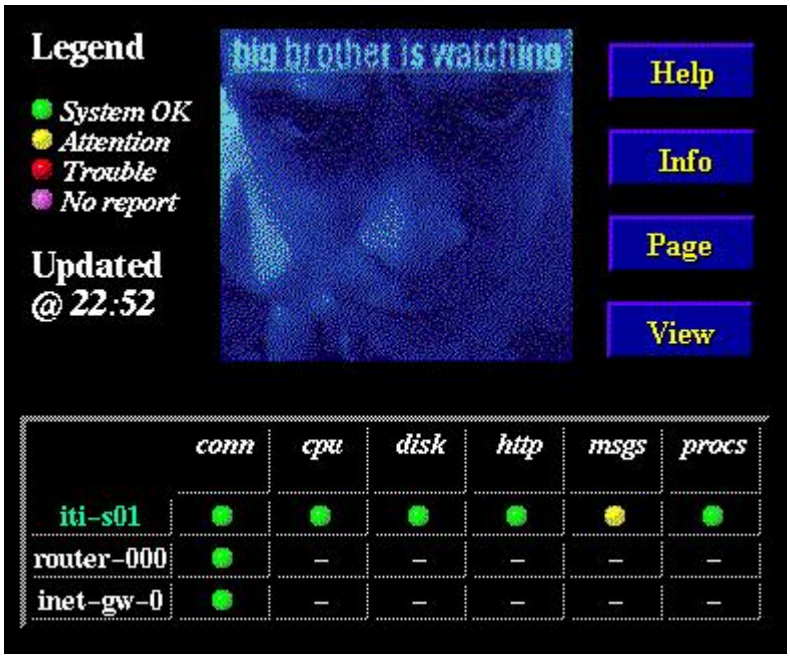


Figure 2

```
      Legend           [BIG BROTHER IMAGE]   [help]
          [grn] System OK  [BIG BROTHER IMAGE]   [info]
          [yel] Attention  [BIG BROTHER IMAGE]   [page]
          [red] Trouble    [BIG BROTHER IMAGE]   [view]
          [blu] No report  [BIG BROTHER IMAGE]
          Updated @ 22:52  [BIG BROTHER IMAGE]
                 conn    cpu    disk    http    msgs    procs
       iti-s01  [grn]   [grn]  [grn]   [grn]   [yel]    [grn]
    route-r-000 [grn]     -      -
-       -        -
     inet-gw-0 [grn]     -      -
-       -        -
```

As you can see, Big Brother is watching. While enduring the scrutiny of the Orwellian face peering out at me, I examined the rest of the display. It is colored like a traffic signal (green/yellow/red), and the update time is clearly displayed beneath it. To the right of “Big Brother” are four buttons, marked clearly **Help**, **Info**, **Page** and **View**. Beneath the header area is a table with six column headings and three rows, each neatly labelled with a computer host name. The boxes formed by the intersection of the rows and columns contain

attractive green and yellow balls. The overall effect is like a decorated tree. The left side of the screen has a yellow tint, gradually becoming black at the center.

Selecting the **Help** button gives a brief explanation of Big Brother. Choosing the **Info** Button provides a much longer and more detailed explanation of the system, including a graphic that really *is* worth a thousand words. The **Page** button sends a signal to a radio-linked pager—not at all what I had expected. Finally, the **View** selection provides a brief but perhaps more useful view of the information, isolating only the systems with problems.

In my case, only the “iti-s01” system was displayed. My browser cursor indicated a link as it passed over each colored dot, so I clicked on the blinking yellow dot and received this message:

```
yellow Tue Feb 18 22:50:53 EST 1997 Feb 16 12:22:33
iti-s01 kernel: WARNING: / was not properly dismounted
```

This puzzled me at first. How on earth could it know that? It turns out that Big Brother (BB) checks the system /var/log/messages file periodically and alerts on any line that begins with either WARNING or NOTICE. As I am certain Sean MacGuire is very conscientious, I suspect he adds that line to his message file, so the viewer can see how Big Brother reports its findings.

Suddenly, my screen spontaneously updated. The update time had changed by five minutes, and a blinking yellow dot appeared under the column labelled **procs**. I clicked on the blinking yellow dot and was informed that the **sendmail** process was not running. This got me really interested—Big Brother can monitor whether selected processes are running.

Being a little puzzled about the screen's ability to update itself, I viewed the document source and discovered some HTML commands that were new to me:

```
<META HTTP-EQUIV="REFRESH" CONTENT="120">
<META HTTP-EQUIV="EXPIRES"
CONTENT="Tue Feb 18 23:22:07 CST 1997">
```

The first **META** line instructs browsers to get an update every 120 seconds. The second tells the browser to get a new copy after the expiration time and date—very clever.

I returned to the graphics window and discovered that the yellow area on the left had changed to red. A new host name row appeared with a blinking red dot under the column labelled **conn**. I clicked on the blinking red dot and read this message:

```
red Tue Feb 18 22:59:11 CST 1997 bb-network.sh:
Can't connect to router''000... (paging)
```

The connection to the machine called **router-000** had been interrupted, and the administrator had been paged. Amazingly, while in Texas, I had become aware of a network outage in Montréal, Canada. This really had possibilities—perhaps someday I may get to take a vacation.

Big Brother Installation

I was so impressed with Big Brother that I decided to use it. Sean has thoughtfully made its acquisition easy, but requests that you fill out an on-line registration form with your name and e-mail address. He also likes to know where you heard about Big Brother. I filled out his forms in early November 1996, and received an e-mail survey form in late December. To download Big Brother and to get technical information about how the system works and how to install and configure the package, go to http://www.iti.qc.ca/iti/users/sean/bb-dnld/bb-dnld.html.

When I clicked on the link to download Big Brother, I ended up with a file called bb-src.tgz. I impetuously gunzipped this to get bb-src.tar. I then thought better of the impending error of my ways and decided to download and print the installation instructions before going further. Installation procedures for Big Brother can be found at http://www.iti.qc.ca/iti/users/sean/bb-dnld/bb-install.html, as well as other information about how to set up the system. Just in case, I also grabbed and printed the debugging information (as it turned out, I did not need it) provided at http://www.iti.qc.ca/iti/users/sean/bb-dnld/bb-debug.html.

I had no problems following the installation instructions. I decided to make the $BBHOME directory /usr/src/bb. The automatic configuration routines are said to work for AIX, FreeBSD, HPUX 10, Irix, Linux, NetBSD, OSF, Red Hat Linux, SCO, SCO 3/5, Solaris, SunOS4.1 and UnixWare. I can vouch for Linux, Red Hat Linux, Solaris and SunOS 4.1. The C programs compiled without incident, and the installation went smoothly. As always, your mileage may vary. In less than an hour, I was looking at Big Brother's display of colored lights.

At this point, it's a good idea to re-examine the documentation and information files. Personalize your installation as desired, and above all, have fun.

Hacking Big Brother

I admit it. I am a closet hacker. I saw many things about the stock BB distribution that I wanted to improve. Big Brother's modular and elegantly simple construction makes it a joy to modify as desired. The shell scripts are portable, simple, well documented and easy to understand. The use of the modified hosts file to determine which hosts to monitor was gratifyingly familiar. The **bbclient** script made it extremely easy to move the required

components to another similar Unix host. Sean has done a remarkable job in making this package easy to install.

I became obsessive-compulsive about hacking BB and modified it slightly, working from Sean MacGuire's v1.03 distribution as a base. I forwarded my changes to him for possible inclusion in a later distribution.

Features I added to BB proper include:

- Links to the info files in the brief view (bb2.html), where I needed them most.
- Links to html info files for each column heading and the column info files themselves. I placed these files in the html directory along with bb.html and bb2.html, and gave them boring names like conn.html, cpu.html, ... smtp.html.
- Checks to determine if ftp servers, pop3 post offices and SMTP Mail Transfer Agents (MTAs) are accessible ($BBHOME/bin/bb-network.sh). These checks all use **bbnet** to **telnet** to the respective ports. I followed Sean's style of adding comments to the bb-hosts file as follows:

```
128.194.44.99   behemoth.tamu.edu  # BBPAGER smtp ftp pop3
165.91.132.4    bryan-ctr.tamu.edu   # pop3 smtp
128.194.147.128 csdl.tamu.edu    # http://csdl.tamu.edu/ ftp smtp
```

- Some environment variables to $BBHOME/etc/bbdef.sh for the added monitoring as follows:

```
#
# WARNING AND PANIC LEVELS FOR DIFFERENT
# THINGS. SEASON TO TASTE
#
DFPAGE=Y          # PAGE ON DISK FULL (Y/N)
CPUPAGE=Y         # PAGE FOR CPU Y/N
TELNETPAGE=Y      # PAGE ON TELNET FAILURE?
HTTPPAGE=Y        # PAGE ON HTTP FAILURE?
FTPPAGE=Y         # PAGE ON FTPD FAILURE?
POP3PAGE=Y        # PAGE ON POP3 PO FAILURE?
SMTPPAGE=Y        # PAGE ON SMTP MTA FAILURE?
export DFPAGE CPUPAGE TELNETPAGE HTTPPAGE\
        FTPPAGE POP3PAGE SMTPPAGE
```

- Updated the bb-info.html and bb-help.html pages to reflect a version of 1.03a and a date of 10 February 1997. I also modified them to add brief mention of the new ftp, pop3 and smtp monitoring checks. Specifically, I changed the bb-help.html file to add new pager codes as follows:

1. 100—Disk Error. Disk is over 95% full...
2. 200—CPU Error. CPU load average is unacceptably high.
3. 300—Process Error. An important process has died.
4. 400—Message file contains a serious error.
5. 500—Network error, can't connect to that IP address.

6. 600—Web server HTTP error—server is down.
7. 610—Ftp server error—server is down.
8. 620—POP3 server error—PopMail Post Office is down.
9. 630—SMTP MTA error—SMTP Mail Host is down.
10. 911—User Page. Message is phone number to call back.

- Added sections to the bb-info.html file to explain the ftp, pop3 and smtp monitoring.
- Used a standard tag-line file on each html page that identifies the author and location of the page. Thus, mkbb.sh and mkbb2.sh now look for an optional tag-line file to incorporate into the html documents that they generate. The optional files are named mkbb.tag (for mkbb.sh) and mkbb2.tag (for mkbb2.sh). The shell scripts look for the optional tag-line files in the $BBHOME/web directory, which is also where the mkbb.sh and mkbb2.sh files reside.
- Went through ALL of the html-generating scripts and html files to ensure that they actually had sections and properly placed double quotes around the various arguments.
- Edited the files so that, for the most part, everything fits on an 80-column screen.
- Modified $BBHOME/etc/bbsys.sh to make it easier to ignore certain disk volumes as follows:

```
# DISK INFORMATION
#
DFSORT="4"     # % COLUMN - 1
DFUSE="^/dev"  # PATTERN FOR LINES TO INCLUDE
DFEXCLUDE="-->E dos|cdrom"
                          # PATTERN FOR LINES TO EXCLUDE
```

- I modified $BBHOME/etc/bbsys.linux, so that the **ping** program is properly found, as follows:

```
# bbsys.linux
#
# BIG BROTHER
# OPERATING SYSTEM DEPENDENT THINGS
# THAT ARE NEEDED
#
PING="/bin/ping"   # LINUX CONNECTIVITY TEST
PS="/bin/ps -ax"   # LINUX
DF="/bin/df -k"
MSGFILE="/var/adm/messages"
TOUCH="/bin/touch" # SPECIAL TO LINUX
```

- Added the ability to dynamically **traceroute** and **ping** each system being monitored. I spoke with Sean about it, and, in keeping with the KISS (Keep It Simple, Stupid) principle, we thought these features were best added to the info files. The user portion is pretty obvious in the source of the info file. The cgi scripts are very simple shell scripts as shown in Listing 1.

Future Enhancements of Big Brother

Sean MacGuire is the primary author of Big Brother. In the finest tradition of decentralized shared software development, Sean solicits improvements, suggestions and enhancements from all. He then skillfully incorporates them as appropriate into the Big Brother distribution. Thus, like Linux, Big Brother is in a dynamic state of positive evolution with contributions from a cast of thousands (at least dozens). This constrained anarchy produces interesting results with an international flavor.

Jacob Lundqvist of Sweden is actively improving the paging interface. He has done a superb job of enhancing the paging portion, adding support for alphanumeric and SMS pagers. Darren Henderson (Maine, US) added AIX support. David Brandon (Texas, US) added proper IRIX support and Jeff Matson (Minnesota, US) made some IRIX fixes. Richard Dansereau (Canada) ported Big Brother to SCO3 and provided support for other df's. Doug White (Oregon, US) made some paging script bug fixes. Ron Nelson (Minnesota, US) adapted BB to Red Hat Linux. Jac Kersing (Netherlands) made some security enhancements to bbd.c. Alan Cox (Wales) suggested some shell script security modifications. Douwe Dijkstra (Netherlands) provided SCO 5 support. Erik Johannessen (Minnesota, US) survived SunOS 4.1.4 installation. Curtis Olson (Minnesota, US) survived IRIX, Linux and SunOS installations. Gunnar Helliesen (Norway) ported Big Brother to Ultrix, OSF and NetBSD. Josh Wilmes (Missouri, US) added Solaris changes for new **ping** stuff.

Many other unsung heroes around the world are undoubtedly working to enhance BB at this very moment.

I am (ab)using Big Brother in ways not originally envisioned by its creator, Sean MacGuire. Texas Agricultural Extension's networks are wildly heterogeneous mixtures of different operating systems and protocols, rather than a homogeneous Unix-based network. I would like to see Big Brother learn about IPX/SPX protocols for Novell connectivity monitoring. I would also like to see Big Brother data collection modules for Macintosh, Novell, OS/2, Windows 3.1x, Windows'95 and Windows NT. Rewriting Big Brother in Perl might better serve these disparate platforms, if I could only find the time.

Big Brother's Impact at Texas Agricultural Extension Service

We now monitor around 122 hosts. Only 20 are actually Unix-based hosts that run Big Brother's bb program internally. Some 28 are Novell servers, 39 are routers, and the rest are a mixture of Macintosh, OS/2, Windows 3.1x, Windows'95 and Windows NT machines running one or more types of servers (34 FTP or 26 HTTP). We also find it useful to monitor our 31 PopMail post

offices and 43 mail hosts and gateways. We are checking connectivity on three DNS servers as well, since they are mission critical.

Big Brother (or, as I now affectionately refer to it, “Big Bother”) is now alerting us to outages five or more times daily. Typically, the system administrator receives a page. BB's display is checked and the info file is used to **traceroute** and **ping** the offending machine to validate the outage. Many connection outages involve routers, DSU/CSUs and multiplexors as well as the actual host. BB's display allows us to quickly see a pattern that aids in diagnosis. The ability to dynamically **traceroute** and **ping** the host from the html info page also helps to rapidly determine the actual point of failure. If the administrator paged cannot correct the problem, he relays it to the responsible person or agency.

Before we installed Big Brother, we were frequently notified of these failures by frustrated users telephoning us. Now, we are often aware of what has failed before they call. The users are also becoming aware that they can monitor the network through the WWW interface. In many instances, we are able to actually correct the problem before it disturbs our users. It is difficult to accurately measure the time saved, but we estimate that Big Brother has had a net positive effect overall.

We have a machine in a publicly visible area displaying the brief view of Big Brother. The green, yellow, red and blue screen splashes are clearly visible far down the hall, helping our network team to be more aware of problems as they occur. The accessibility of the WWW page has made Big Brother useful even to people at the far ends of our network. Thus, Big Brother has become a helpful member of our network team. Maybe now I'll have time to be bored.

Paul Sittler (p-sittler@tamu.edu) is a human being in the service of Texas Agricultural Extension, a part of the Texas A&M University System. As a human being he is, of course, a skilled tool-maker. He enjoys playing with technology and tries to make it useful to others of his species. He is a shy man of simple tastes, who still has a discriminating palate with respect to German wine. He is multilingual, being at least marginally conversant in several human languages and competent in several computer dialects as well. He was born with a peculiar genetic defect that requires him to disassemble and reassemble things rather than merely use them.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #40, August 1997

Our experts answer your technical questions.

Trouble With crontab

I'm having trouble using the crontab file. This is the first time I have tried it and have not been able to get it working. The daemon **crond** is running. In my crontab file I have the following line:

```
30 6 * * * /home/Talon/automail
```

automail is a shell script to log on to my Internet provider, get my mail and log off. As far as I can tell **crond** never executes this command. —Jeff Largent

Making Programs Executable

Are you sure your **automail** program is executable? If it isn't, type:

```
chmod +x /home/Talon/automail
```

—Pierre Ficheux, Lectra Systèmès pierre@rd.lectra.fr

/dev/fd0 Error Message

When I try to mount /dev/fd0 I get the following error message:

```
/dev/fd0 is not a block device
```

What does that mean and how do I fix it? I can **fdformat** a floppy, but I can't mount or create a file system on one. —Scott Petinga

Identifying Block Devices

Check out the type of /dev/fd0—it should be a block device. If it is properly identified, the leading character for this device in the directory list will be a **b**. For example:

```
$ ls -l /dev/fd0
brwxrwxrwx   1 root    floppy     2,    0 Jul 18  1994 /dev/fd0
```

If the b is not present, re-create the device by typing:

```
rm /dev/fd0
mknod /dev/fd0 b 2 0
```

—Pierre Ficheux, Lectra Systèmès pierre@rd.lectra.fr

Mysterious Zombie Process

Every time I start up the X program **netcfg**, it generates a zombie process. I haven't been able to get rid of it. Nobody in my dorm can figure this out either. —Scott

Bug Fixes in netcfg

netcfg is a python script that Red Hat created to ease the configuration of network parameters. This script has been expanded and had many bugs fixed since 4.1 was released.

I suggest you upgrade to the latest netcfg-rpm (which is netcfg-2.15-1.i386.rpm as of this writing). —Mario de Mello Bittencourt Neto, Argo Internet mneto@buriti.com.br

Limiting Directory Size

How do I limit the size of a directory? I want to limit each customer's directory to 1MB. If a customer attempts to upload 2MB of content via **ftp**, I want Linux to reject the attempt to store more than 1MB. How do I accomplish this disk management feat? —M. Lamberson

Install Quota Support

This is easy. Install the **quota** support that will allow you to control the maximum size of a user's home directory. Assuming that you have FTP configured to redirect non-anonymous FTP logins to the actual home directory, it is simply a matter of configuring **quota** support in your kernel. Just go to /usr/src/linux and type:

```
make menuconfig
```

Under the **Filesystems** entry, check for **quota** support. Recompile the kernel, reboot and you can configure **quota** as you wish. —Mario de Mello Bittencourt Neto, Argo Internet mneto@buriti.com.br

init Run Levels

What do the different run levels listed by the **init** command do? I have looked in various Unix books and also in *Linux Configuration and Installation*, but I can't find any information. —Paul Sutton

Different Levels of Operation

A Unix run level is a state of system configuration. Each level signifies a certain level of operation. The run levels are controlled (and are relevant to) the **init** daemon, which is the first process that is executed when any Unix system is first loaded.

init's job is to start and stop processes, and the run levels help determine which to control. The file /etc/inittab contains entries that **init** uses to decide which processes to start and how to start them. An entry looks like this:

```
NM:LEVs:WHEN_AND_HOW:COMMAND
```

NM *is a two-letter identifier for the command—each command must have its own, unique identifier.* ***LEVs*** *is a list of levels during which to run the command. It is typically a number or set of numbers from 0-5 but can also be an* ***S*** *or even nothing, depending on your version of* ***init****.* ***WHEN_AND_HOW*** *is an option specifying when the command should be run, whether* ***init*** *should wait for it to finish or not, whether it should be restarted when it dies and so on.* ***COMMAND*** *is the full path and file name of the command to execute.*

The run level corresponds to the **LEVs** list. For example, the following lines are in my inittab file:

```
# Start the local dial-in services
s0:45:respawn:/usr/local/sbin/mgetty ttyS0 \
        vt100 -D -x 0
s1:45:respawn:/usr/local/sbin/mgetty ttyS2 \
        vt100 -D -x 0
```

I have defined two processes, ***s0*** *and* ***s1****, which will be executed any time the system is in run levels 4 or 5. The* ***respawn*** *option tells* ***init*** *to re-execute the command when the command terminates. (This allows my dial-in lines to accept the new callers when users hang up.) Finally, you can see the command that is run for each entry.*

By using run levels, a system administrator can configure a system to automatically start and stop certain processes when they put their systems into different run levels. The most common use for this capability is to have a normal, multi-user level and a single-user administration level. By triggering this level, they can have their servers automatically terminate user processes and disable network support in order to administer their systems.

You can get more information on Unix run levels from the **INIT(8)** and **INITTAB(5)** man pages. Since many different Unix flavors exist, there are also many different versions of **init**. So, you should examine your man pages for exact details. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com

Making Sense of Core Dumps

How does one read a core dump file? Occasionally, a machine will crash and a core dump file is output. When I try to read them (using the **more** command) they are full of meaningless characters. I have yet to find anything on how to read these files except for a debugger for debugging the programs that caused the dump—I never know which program caused the core dump. Any ideas on other avenues of determining what happened? —G. Hendricks

Using gcc and gdb

Core dump files are process states for the process that died. When a process terminates with one of various signals (such as **SIGSEGV**, the segment violation, typically indicating a memory-related bug in the program) and the process owner's **ulimit** (see your shell's man page) allows for core files, a core dump will be created. It contains information such as the entire set of memory allocated to the program, where the program was when it died and what it was doing.

A core dump is an invaluable tool to Unix programmers. By using it in conjunction with a debugger, a programmer can see what went wrong with his or her program.

To examine one of these files, you typically need two things. First, the program must be compiled and linked using **gcc** with the **-g** switch set, which instructs the compiler to place debugging information in the executable. Although any program can produce a core file, the core file can only tell a programmer the location in the program where the fault occurred and the values of certain variables if this debugging information is available.

The second tool that is required is the debugger. If you have installed the development kit, chances are you already have this. The standard Linux

debugger is **gdb** (the GNU Debugger) and is part of the gcc development kit. A programmer might then use this command to look at a core file:

```
gdb programname core
```

*Core files are typically useful only to programmers, and a debugger is not a very friendly program (**gdb** is certainly no exception). If you have no programming experience, you will probably not increase your knowledge of what went wrong by examining a core file in this way. —Chad Robinson, BRT Technical Services Corporation chadr@brttech.com*

Can't Receive Mail

I have loaded Linux and have all the settings for a full Internet connection. I can **telnet** to and from my computer and can send mail out. I have not been able to configure the system to receive mail. Any suggestions? —Jay Melton

Adding Sendmail to Startup Scripts

Most likely you just don't have **sendmail** running as a daemon. You can start up **sendmail** as a daemon with a command like:

```
sendmail -bd -q15m
```

If that doesn't cause any odd errors, you'll want to add that command to your startup scripts. Check to make sure /etc/rc.d/init.d/sendmail.init exists. If it does, use the run level editor to make it start in run levels 2, 3 and 5 and stop in run levels 0, 1 and 6. —Steven Pritchard, President Southern Illinois Linux Users Group steve@silug.org

Mirroring Sites

With what package and how can you mirror your favorite software site? —Andreas J. Bathe

Perl Script Mirror

There is a Perl script appropriately called mirror that works great for mirroring ftp sites. It is available from ftp://sunsite.doc.ic.ac.uk/packages/mirror/ and comes with excellent documentation. —Steven Pritchard, President Southern Illinois Linux Users Group steve@silug.org

Archive Index Issue Table of Contents

Advanced search