

Advanced search

Linux Journal Issue #89/September 2001



Features

Passive-Aggressive Resistance: OS Fingerprint Evasion by Rob Beck
Everything you need (including code) to do it.

Taming the Wild Netfilter by David A. Bandel
iptables, ipchains—what's the difference?

An Introduction to OpenSSL Programming, Part I of II by Eric Rescorla
Filling in the gaps of the OpenSSL manual pages.

Indepth

The Lightweight Directory Access Protocol by Sander van Vugt
Lighter data retrieval and an alternative to NIS.

Configuring XDM by Ron Hume
Taking advantage of this vnc alternative.

Toolbox

Take Command Password's Progress by Bruce Byfield

Kernel Korner Loadable Kernel Module Exploits by William C. Benton

At the Forge Introducing Enhydra by Reuven Lerner

Cooking with Linux Brochettes de Sécurité by Marcel Gagné

Paranoid Penguin GPG: The Best Free Crypto You Aren't Using, Part I of II by Michael D. Bauer

Columns

Focus on Software [Security Applications](#) by David A. Bandel
Focus on Embedded Systems [Embedded Linux at JavaOne](#) by Rick Lehrbaum

Linux for Suits [Lessons in Mid-Crash](#) by Doc Searls
[Geek Law A Question of Licenses](#) by Lawrence Rosen

Reviews

[SuSE 7.2 Professional](#) by Don Marti
[Hacking Linux Exposed](#) by Thomas Osterlie
[Jagged Alliance 2 for Linux](#) by J. Neil Doane

Departments

[Letters](#)

[upFRONT](#)

From the Editor [Security Begins with Me](#) by Richard Vernon

[Best of Technical Support](#)

[New Products](#)

[Archive Index](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Passive-Aggressive Resistance: OS Fingerprint Evasion

Rob Beck

Issue #89, September 2001

If we do not wish to fight, we can prevent the enemy from engaging us even though the lines of our encampment be merely traced out on the ground. All we need do is to throw something odd and unaccountable in his way. --Sun Tzu, The Art of War

OS fingerprinting is a process for determining the operating system a remote host computer is running, based on characteristics of the data returned from the remote host. This can be as simple as connecting up to the host and reading a service banner or as complex as statistical analysis of TCP initial sequence numbers and flags. An outsider has the capability to discover general information, such as which operating system a host is running, by searching for OS-specific differences in implementation of the TCP stack. In some cases, these differences can reveal information as detailed as the OS version number and processor architecture.

By pinpointing the exact OS of a host, an attacker can launch an educated and precise attack against a target machine. In a world of buffer overflows, knowing the exact flavor of an OS and architecture could be all the opportunity an attacker needs. By using software such as Netfilter for Linux, an administrator can evade accurate OS-fingerprinting methods and in some cases even manipulate the results gathered by the external force. While these practices should never be considered a sound security solution, sometimes they can deter and even confuse a would-be attacker if the host poses as an obscure network entity.

While fingerprint evasion does offer a nice layer of obscurity as to the actual OS a host is running, it does not in any way secure the host from various vulnerabilities. Security practices and policies attempt to raise the level of skill required to compromise the system, obscurity only attempts to mask the actual target. Even if your system appears to be running Microsoft IIS5 to the world, this won't protect you if you are running a vulnerable version of let's say

sendmail, and some script kiddie's automated scanner attempts to exploit you. Fingerprint evasion is meant to deter attacks, not prevent them.

Techniques for Discovery

Before attempting to dissuade a potential attacker through OS deception, one must familiarize themselves with the tools and methods used in fingerprinting an OS. The term "attacker" here is used loosely and encompasses those who would attempt to fingerprint a host or those who might have the intentions of doing the system harm. Security has been, and in the writer's mind always will be, a sequence of measure and counter-measure scenarios. By becoming accustomed to the tools and methodologies available for this type of attack you cannot only prepare and plan for current engagements, but also gain insight into what the future may have in store.

Several publicly available tools exist for attempting to fingerprint an OS. Of these tools, one seems to be the popular choice: nmap (www.insecure.org/nmap/index.html) by Fyodor of Insecure.org. **nmap** uses several techniques for attempting to determine the host operating system from a network level, some of them primitive in their approach and others more complex, requiring a good understanding of the TCP/IP protocol and protocol standards. Of the methods nmap incorporates, some of the most notable are:

- FIN Probing—by sending a packet to an open port on a host with nothing more than the FIN flag set in the packet, an attacker can glean information from certain hosts that respond to the requests. This behavior is non-RFC-compliant so it is a good indicator of OS.
- TCP ISN Sampling—TCP packet ISN (initial sequence number) sampling can be a valuable way to determine and categorize the remote host. By watching the ISNs for patterns an attacker can make an educated guess as to the host OS.
- ICMP Error Messaging—through the use of ICMP (internet control message protocol) error messages, an attacker can find useful information based on the host responses. Particular areas of interest are the checksums, error message echo integrity and TOS (type of service) fields in the reply packets.
- TCP Options—perhaps the most revealing aspect of any TCP stack is how it handles optional TCP flags and data. By making specific requests to a host and varying window scales and segment sizes, one can determine which operating system a host is running based on its willingness to accept or respond with these optional parameters.

While all of these methods of OS fingerprinting are at the packet level, great care should be taken to understand your system at a service level. An attacker

could sort and compare packet structures but will often simply query a web server for the "Server" field in the HTTP response header. Knowing which services readily identify themselves, and more importantly the operating system architecture, will show us other avenues that can be used for remote information retrieval.

Client modesty (or lack thereof) can be an excellent way to glean information from a host as well. Unlike the other options, this process can be completely passive. By watching how a client application presents itself to a service, you can make a reasonable guess at the operating system and possibly the architecture. Of these clients, web browsers, e-mail clients and IRC (internet relay chat) clients are most often the biggest offenders. If we were on IRC and requested a CTCP version from a user, and received the reply of "mIRC32 v5.81 K.Mardam-Bey", we could make an educated guess at this point that the host is running some form of the Windows operating system.

Finally, there is exploit testing. While less tactful, it can nevertheless be useful in discovering the operating system of a host. By initiating a series of OS-specific denial-of-service attacks an outsider can test to see if a host is vulnerable. This can determine which rating system a host is running, usually down to the patch level. The Windows community should be grateful that Fyodor and the other fingerprinting-tool authors didn't decide to incorporate this method into their usual slew of scanning techniques.

Why Fingerprint Evasion?

If you have read up to this point, you are at least no doubt a little curious as to why one would go to the trouble of OS fingerprint evasion. Good question! I think the logic here varies from person to person. Everyone has his or her own reasons for wanting, or not wanting for that matter, operating system obscurity.

For some, the extra layer of obscurity helps them feel fuzzy and warm inside. Like the people who feel the need to remove the issue banner from their Telnet login screens, but resort to Telnet rather than SSH for remote access security (obscure, but technically less secure). For others, the notion of operating system obscurity at the network level allows them to fine-tune and tweak their IDS (intrusion detection system) since they have a fairly good idea not only of what should be coming into their network, but also of what data should be leaving it (obscure, cautious and hopefully secure). Some might even have a need for security, where every network they plug in to is a potential hostile network; and the more obscure their operating system is, the bigger the window of opportunity they have to complete whatever the task is at hand without being noticed (obscure, cautious, secure and probably reading your e-mail). Finally, there are those of us who do it for fun, because we can and

because we get some small kick out of being able to fool the unknown individuals around us who persist on launching scans in our direction (yes, guilty as charged).

Evasion

Now it's time to try our luck at fingerprint evasion. Familiar with some of the common techniques used in determining a host's operating system, we can reverse engineer these concepts to aid us in hiding our operating identities.

First, we need to make sure that all patches are in place and the system is secured. As I stated before, obscurity should only be entertained after security has been implemented. I'm sure some would disagree here, relying solely upon obscurity for their means to a secured system, but what good is obscurity if that script kid33's automated script gains root on your machine tonight? I'm willing to bet once he has root access, what flavor of Linux you are running isn't on his or her list of things to figure out.

Second, we need to observe our services. Do they match up with the operating system we are hoping to pass ourselves off as using? In most cases this isn't as much of a concern since most UNIX environments share similar if not the same services. But if you are hoping to present yourself as a Windows machine, or even a Cisco router for that matter, it may not be to your advantage to show up having IRCd running. Make an effort in matching up your services with a suitable decoy host.

While we're on the subject of services, it is also a good idea to begin grepping through the source code of these services looking for banners or common identifiers of the services. Some subtle identifiers could be the supporting of ASP pages or web content that is served compressed in gzip format. For most people this will be a lot of work. Again, it's up to you to gauge what level of obscurity and conformity with your new decoy host you are trying to achieve.

Next, we need to look at how our host appears on a network vs. how our decoy host should look on a network. To make this a little bit easier I suggest studying already documented materials, namely the current fingerprint files used by the tools themselves. Time should be taken to note not just how your decoy host responds to usual queries, but also what special flags it supports in TCP. TCP flags are useful information for outsiders to determine what OS you are running. The fingerprint files don't include all possible responses a host might give, just simple techniques that work reproducibly. Depending on what level of obscurity you hope to achieve, it may be worth looking into fingerprint information not used by nmap (OSPF, OOB, IPv6, etc.). Or the joy of thoroughness could be outweighed by the sleepless nights you would spend gathering this information.

Finally, a decision needs to be made. Are you crafty, or are you paranoid? If you answer to the latter, then you most likely want to continue by obfuscating your client software. As mentioned above, a host's client software tends to give out all kinds of information regarding the system, either directly or indirectly. In our previous example an IRC client lists itself as being for Win32, but there are also more subtle ways of determining a host, such as reading the e-mail headers of outgoing mail. Once again, it all comes down to how many sleepless nights you are willing to spend before your system meets your criteria.

Potential Problems

OS fingerprint evasion is like any other aspect of security; it takes planning, proper execution and most importantly, understanding. If security policies are not properly implemented, the system could be more vulnerable than if these policies were not implemented at all.

Popularity gives way to recognition. In most realms of software, popularity is a great thing; it brings attention to all your hard work and determination. In the case of OS fingerprint evasion, recognition works against you. If you are using a popular tool or package, eventually vulnerabilities and particulars will be discovered; this is inevitable. These same software-specific identifiers will allow others to fingerprint your counter-measure accurately rather than the operating system itself.

Most every OS attempts to make its TCP ISN sequencing random, in attempt to thwart TCP hijacking and more complex attacks on the system. If your chosen implementation of evasion attempts to alter TCP initial sequence numbers, great care should be taken to ensure you do not downgrade this functionality and put your host at risk to these types of attacks.

As with any software package that makes it onto your system, application security should be a primary concern. Part of the evasion process is masking existing services; the other comes in the form of code, which is meant to filter your traffic and mask what you put on the wire. Great care should be taken to ensure that the application produced for this task is secure through good programming practices and rigorous testing. All it takes is one poorly thought-out `strcpy()` to turn this asset quickly into a liability.

One of the evasion tactics previously listed is to alter the service banners of software that identifies itself. Be careful because some add-on software packages actually use these same banners to determine compatibility with the current system software.

Risk vs. Reward

Having established that evasion does not mean security, we need to look at another aspect of this process, namely performance. Since a good evasion setup filters your traffic en masse, it is feasible that system performance will suffer. Obviously if you have a site that hosts web pages for 10,000 clients, performance is a bigger issue than if you simply have a Linux box set up somewhere for you and your friends to check e-mail and IRC. As an administrator, you need to decide which is the bigger reward for you (and your users), performance or privacy.

Proof of Concept

To illustrate the feasibility and relative ease of fingerprint evasion I have included a small sample user-space application (OSFPE) for Linux, which makes use of the Netfilter kernel modules [see Listing 1 at ftp.linuxjournal.com/pub/lj/listings/issue89/4750.tgz]. Through the use of such software as Netfilter in Linux, OS fingerprint evasion is becoming increasingly more efficient. Similar modifications and applications are sprouting up all over the place; in BSD it is possible to accomplish this task via ipfilter and a moderate amount of code (during the time of this writing ipfilter has been removed from the BSD CVS tree, sorry guys). Windows users (who are by far at the biggest disadvantage in this arena) are discovering ways to shim their TCP/IP communications, and with the inception of Libpcap for Win32, capture and forge their own packet responses.

Netfilter at a Glance

Netfilter, as stated by its author, is "a framework for packet mangling". Sounds fun, eh? Netfilter interfaces with the Linux kernel (kernels 2.4.x and above to be exact) and registers hooks for each protocol. If the proper rules are in place, these hooks capture incoming or outgoing network traffic that match specified rules. These packets are then processed and marked for either NF_DROP to have the packet dropped, NF_ACCEPT to accept the packet for normal processing on the stack or NF_QUEUE to have the packet queued for manipulation in user space. If the packet gets queued for manipulation in user space, the ip_queue driver places it in a queue; it is then handled asynchronously by any applications running in user space that have registered themselves for these types of packets. When these applications pull the packets from the queue they have the ability to manipulate, accept and reject the packets. If the packets are accepted, they are handed off to the next application running that has registered for such a packet. If the packet is flagged for NF_DROP, the packet is dropped and processing of that particular packet ceases. Through the use of Netfilter, applications in user space essentially have kernel-level control of network traffic.

iptables

iptables is an application used to interface with Netfilter to set, view and remove a system's current network filtering rules. I make mention of iptables here because in developing the proof-of-concept code we felt it was a better idea to introduce users to the iptables program for rule administration rather than having the application handle them. This will allow people to better understand what is going on with the packet queuing.

What We Did

By taking advantage of the Netfilter modules and iptables rule administration program we were able to set up rules to capture incoming UDP, TCP and ICMP packets. Based on the incoming packets and the source host we either allow them to access the system normally or craft responses to appear as a Windows host, as defined in one of nmap's OS fingerprint entries. Here is the fingerprint we were attempting to match, and a brief walk-through on how we accomplished this goal:

```
TSeq(Class=TD|RI%gcd=1|2|3|4|5|8|A|14|1E|28|5A%SI=<1F4)
T1(DF=Y%W=2017|16D0|860|869F%ACK=S++%Flags =AS%Ops=M|MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=0%ACK=0%Flags=AR%Ops=)
T4(DF=N%W=0%ACK=S++|0%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=S++|0%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLen=38%RIPTL=148%RID=E%RIPCK=E%UCK =E%ULEN=134%DAT=E)
```

The first line states that we need to build a time dependent (TD) TCP sequencing or one that has random increments (RI) equal to, but not greater than, 0x1F4 (500). This was actually pretty easy to accomplish, or match up I should say. First we grabbed the incoming packet, took the TCP sequence number, generated a pseudo-random number between 1 and 500 and added the values together. This met both the random increment and greatest common denominator (gcd) requirements for the fingerprint.

Next we broke down all the various packet tests (T1-T7) and created cases for them in our TCP handler. All of these are pretty straightforward and simply dictate how the host should respond to different types of packets to open and closed ports, the exact tests and their parameters are covered more in-depth in Fyodor's paper on remote OS detection.

Next we matched up our response for a UDP port-unreachable query. What nmap does here is send a UDP packet to a closed port on the host and wait for a response in the form of an ICMP port-unreachable packet. ICMP port-unreachable packets simply tell the querying host that the port to which they attempt to deliver a UDP message failed because there is no listening UDP service on that port. On some networks these messages never get sent back

and are dropped at the router. In order to conform to the fingerprint we made an effort to send back what they were expecting.

Finally, as an extra little bonus we sent back Syn-Ack packets to the host for specific ports on our host if they happen to scan for these TCP ports as being open. Similarly, we sent back no response for particular UDP ports that we want to appear to be open on our host (as stated above, only closed UDP ports send back a port-unreachable message). When the scan of our host is complete, it should appear as though TCP ports 135 and 139 and UDP ports 135, 137 and 138 are open. If we attempt to fingerprint our host we should match up with the above-listed fingerprint and get the listing as "Windows NT4 / Windows 95 / Windows 98".

As a final note, proof-of-concept code is just that, a little piece of programming used to prove a point. Do yourself a favor and don't run this on a critical device. Open it up, learn from it, modify it, exploit it, but don't depend on it. I've made an attempt to keep the code safe and somewhat readable (arguable), but I can't promise anything.

Acknowledgements

Thanks to Rex Warren for all his hard work aiding me with this paper and the sample code, Fyodor for allowing me to reference all his hard work and for such a great security tool, Dan Kurc for reading over my code and calling that baby ugly (Hey! It's my first C program), Sir Dystic for C-isms and Courtnee.

Resources



Rob Beck is currently a security architect for @stake Security Consulting Services, specializing in Windows OS/application penetration assessment and secure build design.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Taming the Wild Netfilter

David A. Bandel

Issue #89, September 2001

Using Netfilter and ipchains to increase security on home systems.

For those of you who have taken the plunge and upgraded from kernel 2.2.X (or even 2.0.X) to 2.4.X, congratulations. If, like a number of folks, you're running some form of firewall using either ipchains or ipfwadm, your scripts may work fine. But sooner or later you're probably going to want to upgrade.

In the 2.4.X kernels, Rusty Russell, the Linux packet-filter guru, and his crew of coders have implemented Netfilter into the kernel. Netfilter is the replacement for ipchains or ipfwadm. Fortunately, Netfilter permits you to keep using ipchains or ipfwadm until you can come to grips with iptables by adding a compatibility layer via a kernel module that permits these older packet filters to run. But Netfilter has so many exciting new additions, you'll want to convert those rules as soon as possible. One word of caution, though, if you load the ipchains or ipfwadm modules, you can't load ip_tables (and vice versa). So it's all or nothing. After reading this article, however, making the change should be easy.

For those new to packet filtering, ignore the ipchains translations and use the iptables examples. While not all ipchains commands and options will be translated to iptables, this text should provide a good idea about how to construct a packet-filter firewall by translating ipchains commands into iptables commands.

The reason you'll want to upgrade to Netfilter is because it, unlike ipchains or ipfwadm, is stateful. What this means is it can track connections and permit incoming responses to outgoing requests without creating gaping holes in the firewall. The connection tracking opens a specific, temporary hole for responses and only from the contacted server. We'll see how this works later. The drawback is that with connection tracking in use, Netfilter will need to use a little more memory because the connections are tracked in RAM. So your 4MB

386-16 may no longer be up to the job, depending on your filtering requirements.

Background

The actual Netfilter implementation is broken into two parts, the kernel portion known as Netfilter and the userland tool that interfaces with Netfilter and creates the rulesets, iptables. Both are required to implement your packet-filtering firewall.

First, we'll concentrate on the kernel portion. Netfilter includes support for IPv4 and IPv6. It does not, however, filter any other protocols, so your firewall should not run IPX, AppleTalk or any other protocol that might be used to circumvent iptables rules. Similarly, you must not enable the kernel fast-switching option. This item is one of the last ones in the network options section of the kernel's configuration menu. The code permits fast switching at a low level in the IP stack. The Netfilter code resides at a much higher level, so fast switching effectively bypasses Netfilter completely.

Kernel Configuration

In order to get started using Netfilter, you'll need to have your kernel compiled for Netfilter support. Most distributions include this support by default, so a quick test is in order. If you can insert the module `ip_tables`, then you won't need to worry about this section. As root, run the command

```
modprobe ip_tables
```

Then run

```
lsmod | grep ip_tables
```

If `ip_tables` show up, you're in good shape. If not, don't worry, rebuilding a kernel is extremely easy. This text won't cover the complete kernel rebuild process, but many resources are available to help you through this step. If you find you need to rebuild your kernel, the Sidebar will provide you some guidance on what to include for a complete Netfilter-enabled kernel.

Enabling Netfilter in Your Kernel

Netfilter Modules

If you've built and installed all the modules, all modules will auto-install when a rule is entered except `ip_tables`, `ip_nat_ftp` and `ip_conntrack_ftp`. These can be loaded either manually or as part of your iptables startup script.

A full build and installation of Netfilter produces a large number of modules, but most firewalls will only use a few. The modules that aren't loaded aren't taking up memory, so don't worry about what you don't use.

Getting and Installing iptables

Your distribution may have installed iptables, and it almost certainly did if your kernel has Netfilter support. But if you want the very latest, you'll probably have to get it from the Netfilter site. Netfilter is available at netfilter.filewatcher.org. Download and compile it according to the instructions in the INSTALL file. The following instructions assume the kernel sources are in /usr/src/linux. If not, adjust the following instructions appropriately. If you need to run

```
make pending-patches KERNEL_DIR=/usr/src/linux
```

or

```
make patch-o-matic KERNEL_DIR=/usr/src/linux
```

then you'll need to recompile your kernel before continuing. Otherwise, you can ignore these two commands. In general, the patch-o-matic is for users with special needs and is of interest to the average user.

After running

```
make KERNEL_DIR=/usr/src/linux
```

run

```
make install KERNEL_DIR=/usr/src/linux
```

You're now ready to use iptables.

The iptables Command Line

The iptables command line is broken down into as many as six parts. The first part is the iptables command, which will not be discussed further. The second part is the table specification, and the chain name is the third part. The fourth part is the rule specification, which is the part of the command to match against the IP or ICMP header, but also could be a rule number in some incantations. The fifth part is the target, and the sixth part is the target option. The general command line, then, looks like

```
iptables [-t table] -ACDI CHAIN rule-specification & -j TARGET [target option]
```

The above does not hold true for all incantations but is the most common. You will also find the `-L` command useful. It will be covered in this article, along with several other command-line variants.

Tables and Chains

Netfilter has three tables you need to concern yourself with: `filter`, `nat` and `mangle`. This article, whenever it shows an iptables incantation, will always have a table specified. However, if no table is specified, the `filter` table is assumed. Therefore, if you do not specify a table and your rule fails, try putting a table specification in and try again.

Each table has certain chains available to it. User-created chains will belong to one and only one table. You will see that some built-in chains belong to more than one table, but this is only true for built-in chains. You cannot mix chains from other tables within user-created chains.

The `filter` table is the basic packet-filter table and has the built-in chains `INPUT`, `FORWARD` and `OUTPUT`. Rules added to user-created chains created in the `filter` table can only contain targets valid in the `INPUT`, `FORWARD` or `OUTPUT` chains. Packets traversing the `filter` table will pass through one and only one of `INPUT`, `FORWARD` or `OUTPUT`. The `INPUT` chain will only be traversed if the packet's destination is the local system. The `FORWARD` chain will only be traversed if the packet is passing through the local system and bound for another system. And the `OUTPUT` chain is traversed only by packets originating on the local system with an external destination. Only one chain will be traversed by any given packet. This is in contrast to ipchains that always used the `input` and `output` chains and also used the `forward` chain if the packet was passing through.

Note that in the previous paragraph, the iptables chain names were capitalized, while the ipchains chain names were not. This was deliberate and reflects a change in syntax.

The `nat` table performs network address translation. Built-in chains for `nat` are `PREROUTING`, `POSTROUTING` and `OUTPUT`. Each chain permits a particular target. The `PREROUTING` accepts the `DNAT` target, and the remaining chains accept the `SNAT` target. More on these targets shortly.

The `mangle` table is used to change (mangle) information other than the IP address in the header. It can be used to mark the packets, change type of service (TOS) or change time-to-live (ttl) information.

Rules

The rule-specification portion of each iptables command is the heart of the command. By properly crafting your rule, you can select exactly which packets to which a rule should apply. This selection criteria can be as general or as specific as you need. In most cases, you'll want to make sure specific criteria come before more general criteria.

That said, I'll not belabor rules too much here other than to add that multiple options, some of which carry their own options, can be strung together. But you do need to ensure that rules make sense. For example, don't specify an output interface in an input chain or that rule will never match anything. The syntax may allow you to construct impossible rules, but it's not a good idea. If you're in doubt about a particular rule, you can always make the target a log target then send traffic that matches that rule to see if it does in fact trigger. If you need a tool to test your rules, take a look at SendIP (www.earth.li/projectpurple/progs/sendip.html).

Targets

Netfilter has four built-in targets: ACCEPT, DROP, QUEUE and RETURN. The DROP target replaces ipchain's DENY target. All other targets used are based on modules that load as a target. These include REJECT, LOG, MARK, MASQUERADE, MIRROR, REDIRECT and TCPMSS. Terminal targets, such as ACCEPT, DROP, REJECT, MASQUERADE, MIRROR and REDIRECT, terminate a chain. A LOG target does not terminate a chain. LOG also neither ACCEPTS, REJECTS nor DROPS a packet, so the chain continues to be traversed. Thus the ipchains -l option is now another target but a nonterminating one. The rest of the chain will be traversed until it hits the policy rule.

The policy rule is the overall rule for the chain. If your FORWARD chain contains a policy of DROP, and no rules above match in the chain, the packet will terminate when it hits the policy rule. Your policy rule can only be one of the built-in targets. You cannot have REJECT as a policy rule.

Examples

Before we look at examples, let's set some things up. Scripts are nice, and one that runs from rc.local (wherever that is on your system) is always good. So let's write an rc.iptables script as part of our example to implement Netfilter during bootup (see Listing 1). (Note: all iptables rules will start \$IPT and should continue unbroken to the end of the line, i.e., the next command. Rules should not be broken on a line.)

Listing 1. rc.iptables Script

We've set up some variables to start, stopped forwarding traffic through the system, then inserted some modules. The `ip_tables` module allows us to start writing rules. The `ip_nat_ftp` module is necessary if we are using the NAT table to do SNAT or MASQUERADE (covered below) to allow active FTP. The `ip_conntrack_ftp` allows connection tracking of FTP. This module automatically loads the `ip_conntrack` module because it is dependent on `ip_conntrack`. If you don't need or want the `ip_conntrack_ftp` module, but want to make sure your firewall does IP defragmentation (a good idea), you can substitute `ip_conntrack` for `ip_conntrack_ftp`. Let's continue with our script:

```
for i in filter nat mangle
do
$IPT -t $i -F
$IPT -t $i -X
done
```

The above lines will flush (-F) all rules in the chain shown as the argument to -F. Since no chain name was given as an argument, -F will flush all chains. Note that this has to be done for each table, hence the loop. If you aren't using a particular table, you can delete it from the for statement. As each loop starts, you will note that a new module has been loaded: first the `iptables_filter` module, then the `iptables_nat` module and finally the `iptables_mangle` module. If you remove `mangle` from the loop, the `iptables_mangle` module will not be loaded. Unused modules can be removed at your leisure.

Under `ipchains`, you would have used something like

```
ipchains -F -X
```

to accomplish the same thing.

Before we continue, let's assume the following setup: a home user and three systems that access the Internet through our firewall/workstation PC. Access is via dial-up (`ppp0`). If your setup is different, substitute your external interface for `ppp0`. The systems are not offering any services outside their own network, which is on `eth0`. We do, however, want all systems inside to be able to surf the Web. For this first example, we'll assume a dynamic IP from an outside ISP [see Listing 2, `rc.iptables.dynamic`, at ftp.linuxjournal.com/pub/lj/listings/issue89/4815.tgz].

Since our firewall PC is also a workstation, it will be originating and terminating its own traffic. While not a good idea for a firewall (these should be dedicated systems) on a home network, we really don't need a dedicated firewall. With this in mind, remember that one difference between `iptables` and `ipchains` is in the INPUT, FORWARD and OUTPUT chains. In `ipchains`, packets traversing the FORWARD chain came from INPUT and went through FORWARD to OUTPUT, so we could locate our rules in the INPUT chain and be safe for packets going to

the FORWARD chain. The iptables implementation only uses INPUT for the local system and FORWARD for the other systems. In our case we need identical rules in each of the FORWARD and INPUT chains. To prevent duplicating a lot of rules, let's create a user chain called tcprules and call it from both INPUT and FORWARD. Continuing our script then:

```
$IPT -t filter -N tcprules
```

The ipchains equivalent of this rule would be the same, excluding the -t filter option of the incantation:

```
ipchains -N tcprules
```

Now a little Netfilter magic. We want to prevent someone from connecting to our systems from the outside but permit connections to the outside to be established by our users. The following rules make use of Netfilter's stateful capability:

```
$IPT -t filter -A tcprules -i ppp+ -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -t filter -A tcprules -i ! ppp+ -m state --state NEW -j ACCEPT
$IPT -t filter -A tcprules -i ppp+ -m state --state NEW,INVALID -j DROP
```

If you want to do something similar in ipchains, the closest you could come is to deny syn packets to the ppp+ interface:

```
ipchains -A input -i ppp+ ! -y -j DENY
```

A couple of quick notes at this point. First, the "!" negates whatever follows it. So **! ppp+** is the same as specifying all other interfaces (in the case of our home user, lo and eth0). The "+" on the end of the ppp tells Netfilter this rule applies to all ppp interfaces.

As for the ESTABLISHED, RELATED, NEW and INVALID arguments, they are more than they appear to be. ESTABLISHED permits traffic to continue where it has seen traffic before in both directions. ESTABLISHED obviously applies to TCP connections but also to UDP traffic, such as DNS queries and traceroutes as well as ICMP pings. In fact, packets are first checked to see if the connection exists in the connection tracking table (/proc/net/ip_conntrack). If so, the chains aren't run, the original rule is applied and the packets pass. In some cases, Netfilter is faster than its predecessor because of this check. The RELATED argument covers a multitude of sins. This argument is applied to active FTP, which opens a related connection on port 20, but also applies to ICMP traffic related to the TCP connection. The NEW argument applies to packets with only the SYN bit set (and the ACK bit unset). The INVALID applies to packets that have invalid sets of options, as in an XMAS tree scan.

The above rules permit the internal systems to pass new connections internally and externally, but don't permit incoming new or invalid packets.

Since we're going to be masquerading our network behind our firewall, we'll need to set up a masquerading rule. This process is very similar to the one used in ipchains. Assuming our internal network is 192.168.0.0/24, we'll use the following rule:

```
$IPT -t nat -A POSTROUTING -o ppp+ -s 192.168.0.0/24 -d 0/0 -j MASQUERADE
```

One difference ipchains users should note is the use of -o ppp+ instead of -i ppp+. This is because with ipchains, we dealt with interfaces as -i. With iptables, -i stands for input interface, and -o stands for output interface. If you mistakenly use -i ppp+ in the line above, no masquerading will take place. In fact, the rule should never match at all.

Let's finish our script by implementing the tcprules above where they are needed, implementing our filter-table policies and turning ip_forwarding back on:

```
$IPT -t filter -A INPUT -j tcprules
$IPT -t filter -A FORWARD -j tcprules
$IPT -t filter -P INPUT DROP
$IPT -t filter -P FORWARD DROP
echo 1 > /proc/sys/net/ipv4/ip_forward
```

By default, the filter-table policies are all ACCEPT. Given the stateful nature of Netfilter, this affords sufficient protection, unlike ipchains. If you think about the stateful rules we've implemented, you'll see that no harm should come with the default policy; in fact, nothing should ever arrive at the default policy. But some believe that it's always better to play it safe, so we can drop anything not addressed already. Note that the policy doesn't have a target per se, so we don't use -j. This is also why only built-in targets can be policies—policies aren't really targets. If you really want REJECT as your policy, you'll need to add something like the following:

```
$IPT -t filter -A tcprules -i ppp+ -j REJECT --reject-with icmp-host-unreachable
```

The above rule applies to all packets coming in on your ppp interface. Make sure this rule is your last rule because nothing will pass this rule.

While you could make the rule

```
$IPT -f filter -A tcprules -j REJECT --reject-with icmp-host-unreachable
```

you will find that your internal hosts are also blocked because this rule will apply to all interfaces (something you should keep in mind with policies as well).

At this point, I should mention that, unlike ipchains, a target match does not necessarily terminate a chain. For example, if you use the following rule in tcprules:

```
$IPT -t filter -I INPUT -p ICMP -m icmp --icmp-type echo-request -m limit --limit 1/minute -j LOG --lo
```

the next rule (which may or may not also match this packet) will be processed. If a rule match does not drop, reject, accept or queue the packet and is not a return, the chain will continue.

The ipchains crowd will note that under iptables, LOG is a target by itself rather than a simple -l option following the target, as with ipchains. This permits some flexibility, as can be seen from the above rule. The limit match prevents someone malicious from flooding your logs. The LOG target can send a prefix with its message so that it's easy to grep from the logs.

Before we move on, all ipchains users know you can view the chains by using something like

```
ipchains -L -n
```

This will show you the chains. With Netfilter, we need to look at the chains, table by table:

```
iptables -t filter -L -n
iptables -t nat -L -n
iptables -t mangle -L -n
```

You can add a -v to get even more information on each rule:

```
iptables -t filter -L -nv
```

Next is an example to demonstrate the new SNAT and DNAT. If you understand the mangle table and its use, you probably don't need this article and will be e-mailing me about any errors you've noticed.

In this example, we'll assume our home user has a broadband connection and is using eth0 for the internal network and eth1 for the external network. The script begins the same as before, but when we get to the tcprules, we're going to do something a little different. Here, we're going to assume the user has a static IP of 209.127.112.17, which gives us more leeway [see Listing 3, rc.iptables.static, at ftp.linuxjournal.com/pub/lj/listings/issue89/4815.tgz]. In fact, we can also assume the user has his own domain name and is running his own e-mail server on port 25, which is located on an internal system with IP 192.168.0.2 (the firewall is 192.168.0.1). His DNS entry points to 209.127.112.17 as his mail server, as shown in Listing 4.

[Listing 4. DNS Entry for Mail Server](#)

The first two tcprules are the same as in the first example. But before we drop all other connections, we accept connections on port 25. Then, in the nat table, we take the port 25 connection and forward it to another internal host on the same port. You can do this for all your connections. Remember that if you do this forward for DNS, you need to forward UDP as well as TCP. In fact, unless someone outside is going to do zone transfers, you can drop the TCP part and only pass UDP.

Notice that now, instead of using MASQUERADE as a target for outgoing connections, we're using SNAT. In case you wondered, the S stands for source, which is what is being changed. In the case of DNAT, we changed the destination of the packet. The argument --to-source can take a range of IPs, so your firewall can look like several hosts. If you have five usable IPs from your ISP, you can use all five as outgoing. Then you can point different services to different IPs and have up to five systems behind your firewall answering DNS queries, hosting web sites, accepting mail, etc. Netfilter will also allow you to do rudimentary load balancing. When a range of destinations is used with DNAT, the system with the least number of connections (not necessarily the system with the lightest load) gets the connection.

About the only other thing you may want to know to get started is how to increase the maximum number of connections tracked. This number is arrived at by default depending on the amount of RAM your system has. However, the number is conservative and can be increased. You can find this number this way:

```
cat /proc/sys/net/ipv4/ip_conntrack_max
```

On my system with 256MB of RAM, the number is 16376.

Concluding Remarks

Using Netfilter's stateful rules, you can actually increase the security of your home system with less effort by making judicious use of its connection tracking. Many more options are also available to you. This article scratches only the surface. I recommend you make use of Rusty's Unreliable Guides available on the Netfilter site (mentioned earlier).

For home users with simple needs, keep your firewall simple. I do not recommend most firewall tools and scripts because they layer unnecessary complexity into your firewall. If you don't understand a rule, don't implement it. The first three stateful rules (using the -m state rule) will keep you in good stead. If an attacker has already been in and compromised a system, the rules won't help. They also won't protect you against e-mail-based trojans, but they

will protect against direct attacks. I suggest, if you don't use IRC, you log and drop outgoing IRC connections:

```
$IPT -j filter -I tcprules -p tcp --destination-port 6667 -j LOG --log-prefix "IRC attempt "  
$IPT -j filter -I tcprules 2 -p tcp --destination-port 6667 -j DROP
```

Also, if you don't need anyone entering your network, don't open any ports (as we did in our second example). This article did not discuss how to segregate your network properly to isolate internet-accessible systems from trusted internal systems. If you require this level of complexity, and your risk assessment asks for it, it might be time to call for knowledgeable help.



David A. Bandel (dbandel@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of *Que Special Edition: Using Caldera OpenLinux*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

An Introduction to OpenSSL Programming, Part I of II

Eric Rescorla

Issue #89, September 2001

Do you have a burning need to build a simple web client and server pair? Here's why OpenSSL is for you.

The quickest and easiest way to secure a TCP-based network application is with SSL. If you're working in C, your best choice is probably to use OpenSSL (<http://www.openssl.org/>). OpenSSL is a free (BSD-style license) implementation of SSL/TLS based on Eric Young's SSLeay package. Unfortunately, the documentation and sample code distributed with OpenSSL leave something to be desired. Where they exist, the manual pages are pretty good, but they often miss the big picture, as manual pages are intended as a reference, not a tutorial.

The OpenSSL API is vast and complicated, so we won't attempt to provide anything like complete coverage here. Rather, the idea is to teach you enough to work effectively from the manual pages. In this article, the first of two, we will build a simple web client and server pair that demonstrate the basic features of OpenSSL. In the second article, we will introduce a number of advanced features, such as session resumption and client authentication.

I assume that you're already familiar with SSL and HTTP, at least at a conceptual level. If you're not, a good place to start is with the RFCs (see Resources).

For space reasons, this article only includes excerpts from the source code. The complete source code is available in machine-readable format from the author's web site at <http://www.rtfm.com/openssl-examples/>.

Programs

Our client is a simple HTTPS (see RFC 2818) client. It initiates an SSL connection to the server and then transmits an HTTP request over that connection. It then waits for the response from the server and prints it to the screen. This is a

vastly simplified version of the functionality found in programs like `fetch` and `cURL`.

The server program is a simple HTTPS server. It waits for TCP connections from clients. When it accepts one it negotiates an SSL connection. Once the connection is negotiated, it reads the client's HTTP request. It then transmits the HTTP response to the client. Once the response is transmitted it closes the connection.

Our first task is to set up a context object (an `SSL_CTX`). This context object is then used to create a new connection object for each new SSL connection. These connection objects are used to do SSL handshakes, reads and writes.

This approach has two advantages. First, the context object allows many structures to be initialized only once, improving performance. In most applications, every SSL connection will use the same keying material, certificate authority (CA) list, etc. Rather than reloading this material for every connection, we simply load it into the context object at program startup. When we wish to create a new connection, we can simply point that connection to the context object. The second advantage of having a single context object is that it allows multiple SSL connections to share data, such as the SSL session cache used for session resumption. Context initialization consists of four primary tasks, all performed by the `initialize_ctx()` function, shown in Listing 1.

Listing 1. initialize_ctx()

Before OpenSSL can be used for anything, the library as a whole must be initialized. This is accomplished with `SSL_library_init()`, which primarily loads up the algorithms that OpenSSL will be using. If we want good reporting of errors, we also need to load the error strings using `SSL_load_error_strings()`. Otherwise, we won't be able to map OpenSSL errors to strings.

We also create an object to be used as an error-printing context. OpenSSL uses an abstraction called a BIO object for input and output. This allows the programmer to use the same functions for different kinds of I/O channels (sockets, terminal, memory buffers, etc.) merely by using different kinds of BIO objects. In this case we create a BIO object attached to `stderr` to be used for printing errors.

If you're writing a server or a client that is able to perform client authentication, you'll need to load your own public/private key pair and the associated certificate. The certificate is stored in the clear and is loaded together with the CA certificates forming the certificate chain using `SSL_CTX_use_certificate_chain_file()`. `SSL_CTX_use_PrivateKey_file()` is used to

load the private key. For security reasons, the private key is usually encrypted under a password. If it is, the password callback (set using `SSL_CTX_set_default_passwd_cb()`) will be called to obtain the password.

If you're going to be authenticating the host to which you're connected, OpenSSL needs to know what CAs you trust. The `SSL_CTX_load_verify_locations()` call is used to load the CAs.

In order to have good security, SSL needs a good source of strong random numbers. In general, it's the responsibility of the application to supply some seed material for the random number generator (RNG). However, OpenSSL automatically uses `/dev/urandom` to seed the RNG, if it is available. Because `/dev/urandom` is standard on Linux, we don't have to do anything for this, which is convenient because gathering random numbers is tricky and easy to screw up. Note that if you're on a system other than Linux, you may get an error at some point because the random number generator is unseeded. OpenSSL's `rand(3)` manual page provides more information.

The Client

Once the client has initialized the SSL context, it's ready to connect to the server. OpenSSL requires us to create a TCP connection between client and server on our own and then use the TCP socket to create an SSL socket. For convenience, we've isolated the creation of the TCP connection to the `tcp_connect()` function (which is not shown here but is available in the downloadable source).

Once the TCP connection has been created, we create an SSL object to handle the connection. This object needs to be attached to the socket. Note that we don't directly attach the SSL object to the socket. Rather, we create a BIO object using the socket and then attach the SSL object to the BIO.

This abstraction layer allows you to use OpenSSL over channels other than sockets, provided you have an appropriate BIO. For instance, one of the OpenSSL test programs connects an SSL client and server purely through memory buffers. A more practical use would be to support some protocol that can't be accessed via sockets. For instance, you could run SSL over a serial line.

The first step in an SSL connection is to perform the SSL handshake. The handshake authenticates the server (and optionally the client) and establishes the keying material that will be used to protect the rest of the traffic. The `SSL_connect()` call is used to perform the SSL handshake. Because we're using blocking sockets, `SSL_connect()` will not return until the handshake is completed or an error has been detected. `SSL_connect()` returns 1 for success and 0 or negative for an error. This call is shown below:


```
/* Connect the TCP socket*/
sock=tcp_connect(host,port);
/* Connect the SSL socket */
ssl=SSL_new(ctx);
sbio=BIO_new_socket(sock,BIO_NOCLOSE);
SSL_set_bio(ssl,sbio,sbio);
if(SSL_connect(ssl)<=0)
    berr_exit("SSL connect error");
if(require_server_auth)
    check_cert(ssl,host);
```

When we initiate an SSL connection to a server, we need to check the server's certificate chain. OpenSSL does some of the checks for us, but unfortunately others are application specific, and so we have to do those ourselves. The primary test that our sample application does is to check the server identity. This check is performed by the `check_cert` function, shown in Listing 2.

Listing 2. check_cert() Function

Once you've established that the server's certificate chain is valid, you need to verify that the certificate you're looking at matches the identity that you expect the server to have. In most cases, this means that the server's DNS name appears in the certificate, either in the Common Name field of the Subject Name or in a certificate extension. Although each protocol has slightly different rules for verifying the server's identity, RFC 2818 contains the rules for HTTP over SSL/TLS. Following RFC 2818 is generally a good idea unless you have some explicit reason to do otherwise.

Because most certificates still contain the domain name in the Common Name field rather than in an extension, we show only the Common Name check. We simply extract the server's certificate using `SSL_get_peer_certificate()` and then compare the common name to the hostname we're connecting to. If they don't match, something is wrong and we exit.

Before version 0.9.5, OpenSSL was subject to a certificate extension attack. To understand this, consider the case where a server authenticates with a certificate signed by Bob, as shown in Figure 1. Bob isn't one of your CAs, but his certificate is signed by a CA you trust.

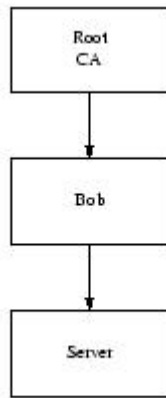


Figure 1. An Extended Certificate Chain

If you accept this certificate you're going to be in a lot of trouble. The fact that the CA signed Bob's certificate means that the CA believes that it has verified Bob's identity, not that Bob can be trusted. If you know that you want to do business with Bob, that's fine, but it's not very useful if you want to do business with Alice, and Bob (of whom you've never heard) is vouching for Alice.

Originally, the only way to protect yourself against this sort of attack was to restrict the length of certificate chains so that you knew that the certificate you were looking at was signed by the CA. The X.509 version 3 contains a way for a CA to label certain certificates as other CAs. This permits a CA to have a single root that then certifies a bunch of subsidiary CAs.

Modern versions of OpenSSL (0.9.5 and later) check these extensions, so you're automatically protected against extension attacks whether or not you check chain length. Versions prior to 0.9.5 do not check the extensions at all, so you have to enforce the chain length if using an older version. 0.9.5 has some problems with checking, so if you're using 0.9.5, you should probably upgrade. The `#ifdef` code in `initialize_ctx()` provides chain-length checking with older versions. We use the `SSL_CTX_set_verify_depth()` to force OpenSSL to check the chain length. In summary, it's highly advisable to upgrade to 0.9.6, particularly because longer (but properly constructed) chains are becoming more popular. The absolute latest (and best) version of OpenSSL is .0.9.66.

We use the code shown in Listing 3 to write the HTTP request. For demonstration purposes, we use a more-or-less hardwired HTTP request found in the `REQUEST_TEMPLATE` variable. Because the machine to which we're connecting may change, we do need to fill in the Host header. This is done using `sprintf()`. We then use `SSL_write()` to send the data to the server. `SSL_write()`'s API is more or less the same as `write()`, except that we pass in the SSL object instead of the file descriptor.

Listing 3. Writing the HTTP Request

Experienced TCP programmers will notice that instead of looping around the write, we throw an error if the return value doesn't equal the value we're trying to write. In blocking mode, `SSL_write()` semantics are all or nothing; the call won't return until all the data is written or an error occurs, whereas `write()` may only write part of the data. The `SSL_MODE_ENABLE_PARTIAL_WRITE` flag (not used here) enables partial writes, in which case you'd need to loop.

In old-style HTTP/1.0, the server transmits its response and then closes the connection. In later versions, persistent connections that allow multiple sequential transactions on the same connection were introduced. For convenience and simplicity we will not use persistent connections. We omit the header that allows them, causing the server to use a connection close to signal the end of the response. Operationally, this means that we can keep reading until we get an end of file, which simplifies matters considerably.

OpenSSL uses the `SSL_read()` API call to read data, as shown in Listing 4. As with `read()`, we simply choose an appropriate-sized buffer and pass it to `SSL_read()`. Note that the buffer size isn't really that important here. The semantics of `SSL_read()`, like the semantics of `read()`, are that it returns the data available, even if it's less than the requested amount. On the other hand, if no data is available, then the call to read blocks.

Listing 4. Reading the Response

The choice of `BUFSIZZ`, then, is basically a performance trade-off. The trade-off is quite different here from when we're simply reading from normal sockets. In that case, each call to `read()` requires a context switch into the kernel. Because context switches are expensive, programmers try to use large buffers to reduce them. However, when we're using SSL the number of calls to `read()`, and hence context switches, is largely determined by the number of records the data was written in rather than the number of calls to `SSL_read()`.

For instance, if the client wrote a 1000-byte record and we call `SSL_read()` in chunks of 1 byte, then the first call to `SSL_read()` will result in the record being read in, and the rest of the calls will just read it out of the SSL buffer. Thus, the choice of buffer size is less significant when we're using SSL than with normal sockets. If the data were written in a series of small records, you might want to read all of them at once with a single call to `read()`. OpenSSL provides a flag, `SSL_CTRL_SET_READ_AHEAD`, that turns on this behavior.

Note the use of the switch on the return value of `SSL_get_error()`. The convention with normal sockets is that any negative number (typically -1) indicates failure, and that one then checks `errno` to determine what actually happened. Obviously `errno` won't work here because that only shows system

errors, and we'd like to be able to act on SSL errors. Also, `errno` requires careful programming in order to be threadsafe.

Instead of `errno`, OpenSSL provides the `SSL_get_error()` call. This call lets us examine the return value and figure out whether an error occurred and what it was. If the return value was positive, we've read some data, and we simply write it to the screen. A real client would parse the HTTP response and either display the data (e.g., a web page) or save it to disk. However, none of this is interesting as far as OpenSSL is concerned, so we won't show any of it here.

If the return value was zero, this does not mean that there was no data available. In that case, we would have blocked, as discussed above. Rather, it means that the socket is closed, and there never will be any data available to read. Thus, we exit the loop.

If the return value was something negative, then some kind of error occurred. There are two kinds of errors we're concerned with: ordinary errors and premature closes. We use the `SSL_get_error()` call to determine which kind of error we have. Error handling in our client is pretty primitive, so with most errors we simply call `berr_exit()` to print an error message and exit. Premature closes have to be handled specially.

TCP uses a FIN segment to indicate that the sender has sent all of its data. SSL version 2 simply allowed either side to send a TCP FIN to terminate the SSL connection. This allowed for a truncation attack; the attacker could make it appear that a message was shorter than it was simply by forging a TCP FIN. Unless the victim had some other way of knowing what message length to expect, he or she would simply believe the length was correct.

In order to prevent this security problem, SSLv3 introduced a "close_notify" alert. The close_notify is an SSL message (and therefore secured) but is not part of the data stream itself and so is not seen by the application. No data may be transmitted after the close_notify is sent.

Thus, when `SSL_read()` returns 0 to indicate that the socket has been closed, this really means that the close_notify has been received. If the client receives a FIN before receiving a close_notify, `SSL_read()` will return with an error. This condition is called a premature close.

A naïve client might decide to report an error and exit whenever it received a premature close. This is the behavior that is implied by the SSLv3 specification. Unfortunately, sending premature closes is a rather common error, particularly common with clients. Thus, unless you want to be reporting errors all the time,

you often have to ignore premature closes. Our code splits the difference. It reports the premature close on `stderr` but doesn't exit with an error.

If we read the response without any errors, then we need to send our own `close_notify` to the server. This is done using the `SSL_shutdown()` API call. We'll cover `SSL_shutdown()` more completely when we talk about the server, but the general idea is simple: it returns 1 for a complete shutdown, 0 for an incomplete shutdown and -1 for an error. Since we've already received the server's `close_notify`, about the only thing that can go wrong is that we have trouble sending our `close_notify`. Otherwise `SSL_shutdown()` will succeed (returning 1).

Finally, we need to destroy the various objects we've allocated. Since this program is about to exit, thus freeing the objects, this isn't strictly necessary, but it would be in a more general program.

Server

Our web server is mainly a mirror of the client but with a few twists. First, we `fork()` in order to let the server handle multiple clients. Second, we use OpenSSL's BIO APIs to read the client's request one line at a time, as well as to do buffered writes to the client. Finally, the server closure sequence is more complicated.

On Linux, the simplest way to write a server that can handle multiple clients is to create a new server process for each client that connects. We do that by calling `fork()` after `accept()` returns. Each new process executes independently and just exits when it's finished serving the client. Although this approach can be quite slow on busy web servers it's perfectly acceptable here. The main server accept loop is shown in Listing 5.

Listing 5. Server Accept Loop

After forking and creating the SSL object, the server calls `SSL_accept()`, which causes OpenSSL to perform the server side of the SSL handshake. As with `SSL_connect()`, because we're using blocking sockets, `SSL_accept()` will block until the entire handshake has completed. Thus, the only situation in which `SSL_accept()` will return is when the handshake has completed or an error has been detected. `SSL_accept()` returns 1 for success and 0 or negative for error.

OpenSSL's BIO objects are stackable to some extent. Thus, we can wrap an SSL object in a BIO (the `ssl_bio` object) and then wrap that BIO in a buffered BIO object, as shown below:

```
io=BIO_new(BIO_f_buffer());
ssl_bio=BIO_new(BIO_f_ssl());
BIO_set_ssl(ssl_bio,ssl,BIO_CLOSE);
BIO_push(io,ssl_bio);
```

This allows us to perform buffered reads and writes on the SSL connection by using the `BIO_*` functions on the new `io` object. At this point you might ask, why is this good? Primarily, it's a matter of programming convenience. It lets the programmer work in natural units (lines and characters) rather than SSL records.

Request

An HTTP request consists of a request line followed by a bunch of header lines and an optional body. The end of the header lines is indicated by a blank line (i.e., a pair of CRLFs, though sometimes broken clients will send a pair of LFs instead). The most convenient way to read the request line and headers is to read one line at a time until you see a blank line. We can do this using the `OpenSSL_BIO_gets()` call, as shown in Listing 6.

Listing 6. Reading the Request

The `BIO_gets()` call behaves analogously to the `stdio fgets()` call. It takes an arbitrary-sized buffer and a length and reads a line from the SSL connection into that buffer. The result is always null terminated (but includes the terminating LF). Thus, we simply read one line at a time until we get a line that consists of simply an LF or a CRLF.

Because we use a fixed-length buffer, it is possible, though unlikely, that we will get a line that's too long. In that event, the long line will be split over two lines. In the extremely unlikely event that the split happens right before the CRLF, the next line we read will consist of only the CRLF from the previous line. In this case we'll be fooled into thinking that the headers have finished prematurely. A real web server would check for this case, but it's not worth doing here. Note that no matter what the incoming line length is, there's no chance of a buffer overrun. All that can happen is that we'll misparse the headers.

Note that we really don't do anything with the HTTP request. We just read it and discard it. A real implementation would read the request line and the headers, figure out if there was a body and read that too.

The next step is to write the HTTP response and close the connection:

```
if((r=BIO_puts
    (io,"HTTP/1.0 200 OK\r\n"))<0)
    err_exit("Write error");
if((r=BIO_puts
    (io,"Server: EKServer\r\n\r\n"))<0)
    err_exit("Write error");
```

```
if((r=BIO_puts
    (io,"Server test page\\r\\n"))<0)
    err_exit("Write error");
if((r=BIO_flush(io))<0)
    err_exit("Error flushing BIO");
```

Note that we're using `BIO_puts()` instead of `SSL_write()`. This allows us to write the response one line at a time but have the entire response written as a single SSL record. This is important because the cost of preparing an SSL record for transmission (computing the integrity check and encrypting it) is quite significant. Thus, it's a good idea to make the records as large as possible.

It's worth noting a couple of subtleties about using this kind of buffered write. First, you need to flush the buffer before you close. The SSL object has no knowledge that you've layered a BIO on top of it, so if you destroy the SSL connection you'll just leave the last chunk of data sitting in the buffer. The `BIO_flush()` call takes care of this. Also, by default, OpenSSL uses a 1024-byte buffer size for buffered BIOs. Because SSL records can be up to 16KB long, using a 1024-byte buffer can cause excessive fragmentation (and hence lower performance). You can use the `BIO_ctrl()` API to increase the buffer size.

Once we've finished transmitting the response, we need to send our `close_notify`. As before, this is done using `SSL_shutdown()`. Unfortunately, things get a bit trickier when the server closes first. Our first call to `SSL_shutdown()` sends the `close_notify` but doesn't look for it on the other side. Thus, it returns immediately but with a value of 0, indicating that the closure sequence isn't finished. It's then the application's responsibility to call `SSL_shutdown()` again.

It's possible to have two attitudes here. We could decide we've seen all of the HTTP request that we care about. We're not interested in anything else. Hence, we don't care whether the client sends a `close_notify` or not. Alternatively, we strictly obey the protocol and expect others to as well. Thus, we require a `close_notify`.

If we have the first attitude then life is simple. We call `SSL_shutdown()` to send our `close_notify` and then exit right away, regardless of whether the client has sent one or not. If we take the second attitude (which our sample server does), then life is more complicated because clients often don't behave correctly.

The first problem we face is that clients often don't send `close_notify`s all. In fact, some clients close the connection as soon as they've read the entire HTTP response (some versions of IE do this). When we send our `close_notify`, the other side may send a TCP RST segment, in which case the program will catch a `SIGPIPE`. We install a dummy `SIGPIPE` handler in `initialize_ctx()` to protect against this problem.

The second problem we face is that the client may not send a `close_notify` immediately in response to our `close_notify`. Some versions of Netscape require you to send a TCP FIN first. Thus, we call `shutdown(s,1)` before we call `SSL_shutdown()` the second time. When called with a “how” argument of 1, `shutdown()` sends a FIN but leaves the socket open for reading. The code to do the server shutdown is shown in Listing 7.

Listing 7. Calling `SSL_shutdown()`

What's Missing

In this article, we've only scratched the surface of the issues involved with using OpenSSL. Here's a (nonexhaustive) list of additional issues.

A more sophisticated approach to checking server certificates against the server hostname is to use the X.509 `subjectAltName` extension. In order to make this check, you would need to extract this extension from the certificate and then check it against the hostname. Additionally, it would be nice to be able to check hostnames against wild-carded names in certificates.

Note that these applications handle errors simply by exiting with an error. A real application would, of course, be able to recognize errors and signal them to the user or some audit log rather than just exiting.

In the next article, we'll be discussing a number of advanced OpenSSL features, including session resumption, multiplexed and nonblocking I/O and client authentication.

Acknowledgements

Thanks to Lisa Dusseault, Steve Henson, Lutz Jaenicke and Ben Laurie for help with OpenSSL and review of this article.

Resources



Eric Rescorla has been working in internet security since 1993. He is the author of *SSL and TLS: Designing and Building Secure Systems* (Addison-Wesley 2000).

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

InDepth

Sander van Vugt

Issue #89, September 2001

Authenticate users on your Linux system by the numbers with the versatile LDAP.

On the Internet, one can find a mass of information that is often organized into very different kinds of directories. The first attempt to establish a standard for directories was made by X.500, an address book designed to provide mail addresses for the X.400 mail application. To retrieve data from this directory, the directory access protocol (DAP) was used, but it was something of a heavyweight.

Other directories were later developed, and although often quite different, most had in common a base in the X.500 standard. There also came a new protocol to retrieve data, the lightweight directory access protocol (LDAP), which nowadays can be used on almost any computer platform to get data out of almost every X.500-compatible directory.

The use of this lightweight directory access protocol will be examined in this article. First we'll take a look at the composition of the directory from which one can access data using LDAP, then we will take a look at how this protocol can be used for Linux. Finally, I will explain how you can even use an LDAP directory to authenticate users on your system, which can be an interesting alternative to network information service (NIS).

Terminology

LDAP can be used to get information from an X.500-compatible directory. This directory is a hierarchically organized database in which different kinds of data are made accessible. Often it is used to store names and mail addresses, but you can also keep information about resources on the network or everything needed to authenticate users on your Linux computers. The hierarchy of the directory is created by the use of container objects, which are also referred to

as directory components (DCs). These DCs can be compared to the domains used in the DNS hierarchy. The LDAP containers even can be linked to DNS domains; so if your company has a registered DNS domain name, like azlan.com, you can make LDAP organizational units to obtain names, such as ou=training, dc=azlan, dc=com.

In the container objects are the leaf objects, also referred to as entries because they are actually nothing more than entries in the LDAP database. An example of such a leaf object is a user with his or her associated mail address and, if you like, all the other information needed to authenticate this user on your system. Each of these entries has a unique name, called the distinguished name (DN). For example, the user Paul in the training department at Azlan, which has a registered .com domain, gets the distinguished name of cn=Paul, ou=training, dc=azlan, dc=com. Besides that, the entry has a common name (CN), a unique identifier of the object in its container; you can compare that to a person's surname.

All these objects have attributes that define the information associated with the object; a user object, for example, can have such attributes as an e-mail address and a password. If you want to use LDAP to authenticate Linux users, it is important that these attributes have the proper values. That is, you need an attribute that represents the UID field from /etc/passwd if you want to be able to use your Linux resources. The exact definition of the entries, the place where they can occur in the directory and the attributes associated with them, is done in the schema. In Linux, the entries are defined in the file slapd.oc.conf and the attributes in slapd.at.conf.

The common data format used to get information into the LDAP database is the LDAP data interchange format (LDIF). If you want to use it, it is very important that mandatory attributes for each of the entries are specified; if not, you get some nasty error codes when you try to define the objects. Mandatory attributes are defined in spad.oc.conf.

OpenLDAP

In Linux, probably the most used LDAP implementation is OpenLDAP (www.openldap.org). Proprietary directories are available that are LDAP-compatible, such as Novell's eDirectory or Netscape's directory. After installation of OpenLDAP on Linux, (often part of a default server installation) some files are copied to your system. Before we take a look at the actual configuration, let's get an overview of the different files copied.

The most important program file that is part of your LDAP installation is the daemon slapd, the standalone LDAP daemon. This is the process you need to activate if you want all the goods of LDAP on your system. If you use more than

one LDAP server on your network, and you want to replicate data between these servers, you also need slurpd. **slurpd** is the process that replicates data from an LDAP master server to one or more LDAP slave servers.

To configure your LDAP server, of course, you need to edit some configuration files. Most of them are in `/etc/openldap`, but pay attention. Sometimes the same files are also in other directories, such as `/etc`, which can make it hard to get a good configuration. If the files in `/etc/openldap` exist in more than one location, I personally prefer to keep only the files in `/etc/openldap` and make links to them at all other locations.

The most important configuration file is `slapd.conf`; you can specify almost all the behavior of `slapd` in it, and it is the file you want to edit before you run `slapd`. Besides `slapd.conf`, there are two files in which the schema is kept: `slapd.oc.conf` and `slapd.at.conf`. In most cases you don't need to do anything with these; they are good as they are. In some cases, however, you do need to edit them in order for an application to work. The last configuration file is `ldap.conf`; it is a small but important file used on an LDAP client to identify the server from which the client needs to get its data.

Besides these configuration files, there are some commands you can use to get data in your directory and to test if there's actually anything in it. **ldapadd** is used to add data to the directory, `ldapmodify` to modify attributes of existing entries and `ldapsearch` to find some specific entries. With these and a few other commands, you can manipulate data in your directory by means of LDIF files.

The last files we'll talk about are the modules that are used if you want to get your system to authenticate on an LDAP server. They are part of other software packages, so they are not always installed on your system. The module that's used by the `nameservice` switch is called `nss_ldap`. You need to specify in `/etc/nsswitch.conf` that you want to retrieve information from an LDAP directory instead of, for example, `/etc/passwd`. Another important module is `pam_ldap`. This is the module you need in Linux to let your users authenticate on an LDAP database, by means of the pluggable authentication modules (PAM). In fact, it's not difficult to configure an LDAP directory on your system. Only four steps are needed to complete setup.

1. Install the Software

If LDAP isn't installed as part of the default server installation on your system, you can download and install it from <http://www.openldap.org/> or one of its mirror sites. To accomplish the installation, first untar the tarball with something like:

```
tar -zxvf openldap-stable-xxxxx.tgz
```

where xxxxx is the version number of the file you downloaded. Then activate the directory created in the preceding step, and run the configure script in this directory. This script verifies that all conditions to install LDAP on your system have been met. Run **make**; first you make the dependencies with **make depend**, then you compile the program with just **make**. Now you can see if everything compiled correctly by running make in the directory /test. Finally, you can actually install the software on your system; type **make install** in the directory created when you expanded the tarball.

2. Edit the Configuration File slapd.conf

After the installation, you'll find an example of the configuration file slapd.conf in the directory /etc/openldap. You will need to edit it to meet the requirements of your organization. For starters, you don't need to make it too complicated; edit the example file to look something like Listing 1.

Listing 1. Example of slapd.conf

Let's take a look at the most important lines in this file. The first two lines are used to include two extra configuration files. In this case, they are the schema files, which are not modified at all, but you do need to instruct slapd as to where it can find them. The line "schemacheck = off" is also not too exciting; it tells slapd that it doesn't have to check the schema. After that there are another two lines that point to some extra files: slapd.pid, which keeps the PID slapd is using, and slapd.args, which keeps the arguments that were used when slapd was started. Then there's the line that defines the kind of database you are using. You can specify ldbm, shell and passwd, but ldbm is the most common.

Then there are three important lines. The first begins "suffix dc=azlan, dc=com"; this line defines the standard container in which slapd should work. In my case, it is equal to the DNS name of the company I work for. Then the name of the account allowed to manage or make modifications to the database is mentioned by its full distinguished name. The third line defines the password used by this manager; as you see, it is written as a plain text password, which isn't very secure, but we'll talk about that later.

The line "directory /usr/local/var/openldap-ldbm" defines the location of the directory where the LDAP database will be installed. Make sure that its mode is 700 and that the owner of the slapd process can read and write to it.

After these lines there are some options that aren't strictly necessary but can be very handy. First is "lastmod on", an option that keeps track of the users that make modification to objects. For that, the attributes modifiersName, modifyTimestamp, creatorsName and createTimestamp are used. Then we

have options that do some indexing. Unfortunately, OpenLDAP isn't the fastest LDAP directory around, so it could be worthwhile to speed things up with some indexfiles. `loglevel 64`, which accomplishes extensive logging, is good if you want to make things work more quickly. The minimal value for this parameter is 1, the maximum is 256 and between that you can use 2, 4, 8, 16, 32, 64 and 128.

Lastly, there is the specification of some access rights to the directory. The default is "read", meaning anyone can read anything, including the passwords. The four lines starting with "access to attr=userpassword" contain the specifications for who can do what with the passwords in the directory. The first line specifies that everyone is allowed to modify his or her own password. Root is allowed to write to any password, while normal users can read but not write to passwords (necessary, of course, to be able to log in to the system).

3. Start slapd

Once you've edited `slapd.conf` to your satisfaction, the next step is to start the LDAP daemon `slapd`. Of course, you could type `slapd` to do that, but you can also tell it to show you all debug messages by adding the option `dn`, in which `n` represents the number of the debug level you'd like to see.

4. Add Data to the Directory

Now you can go on to the next step, which is to add data to the directory. In this example, we'll add some simple data. To do this, you have to compose an LDIF file that could have the contents shown in Listing 2.

Listing 2. Sample LDIF File

If you've made a file like Listing 2, which could be called `~/users.ldif`, you can add it to the directory with

```
ldapadd -D "cn=manager, dc=azlan, dc=com" -W < ~/users.ldif
```

You will be prompted for your password, which is, of course, the password of the root account as specified in `slapd.conf`. If everything goes well, the data should now be added to the directory.

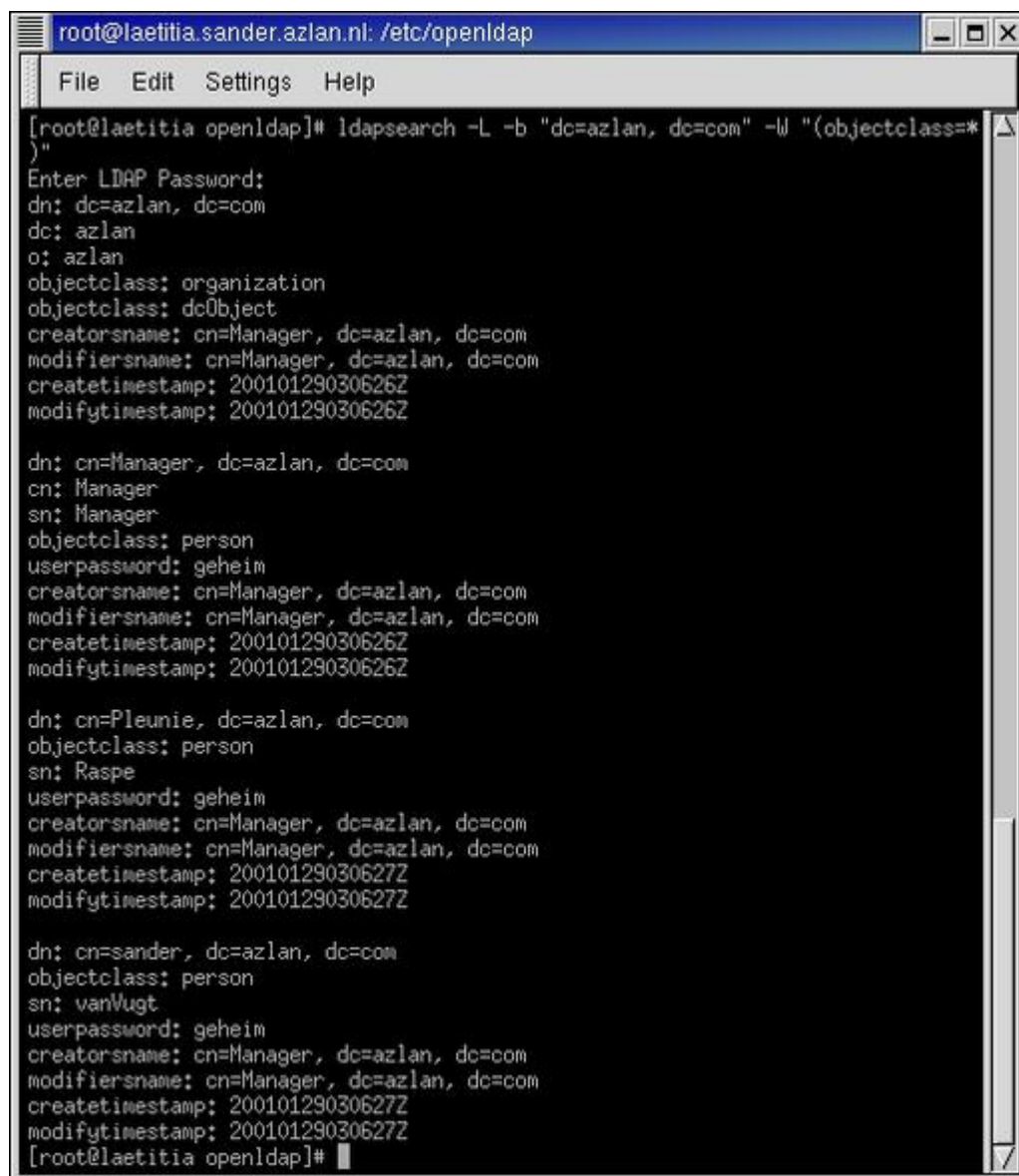
Many errors can be solved by verifying whether `slapd` is really running (oh yeah, it happens) and whether you have any extra spaces in your configuration or LDIF files.

5. See If It Works

Now that you've added the data to the directory, you can actually determine whether it works with the command

```
ldapsearch -L -b "dc=azlan, dc=com" -w "(objectclass=*)"
```

You should get all data in the directory returned as a result (see Figure 1).

A terminal window titled 'root@laetitia.sander.azlan.nl: /etc/openldap' showing the execution of the 'ldapsearch' command. The command is '[root@laetitia openldap]# ldapsearch -L -b "dc=azlan, dc=com" -w "(objectclass=*)"'. The output lists LDAP entries for 'dc=azlan, dc=com', 'cn=Manager, dc=azlan, dc=com', 'cn=Pleunie, dc=azlan, dc=com', and 'cn=sander, dc=azlan, dc=com', including details like objectclass, creator, and timestamps.

```
root@laetitia.sander.azlan.nl: /etc/openldap
File Edit Settings Help
[root@laetitia openldap]# ldapsearch -L -b "dc=azlan, dc=com" -w "(objectclass=*)"
Enter LDAP Password:
dn: dc=azlan, dc=com
dc: azlan
o: azlan
objectclass: organization
objectclass: dcObject
creatorsname: cn=Manager, dc=azlan, dc=com
modifiersname: cn=Manager, dc=azlan, dc=com
createtimestamp: 20010129030626Z
modifytimestamp: 20010129030626Z

dn: cn=Manager, dc=azlan, dc=com
cn: Manager
sn: Manager
objectclass: person
userpassword: geheim
creatorsname: cn=Manager, dc=azlan, dc=com
modifiersname: cn=Manager, dc=azlan, dc=com
createtimestamp: 20010129030627Z
modifytimestamp: 20010129030627Z

dn: cn=Pleunie, dc=azlan, dc=com
objectclass: person
sn: Raspe
userpassword: geheim
creatorsname: cn=Manager, dc=azlan, dc=com
modifiersname: cn=Manager, dc=azlan, dc=com
createtimestamp: 20010129030627Z
modifytimestamp: 20010129030627Z

dn: cn=sander, dc=azlan, dc=com
objectclass: person
sn: vanVugt
userpassword: geheim
creatorsname: cn=Manager, dc=azlan, dc=com
modifiersname: cn=Manager, dc=azlan, dc=com
createtimestamp: 20010129030627Z
modifytimestamp: 20010129030627Z
[root@laetitia openldap]#
```

Figure 1. ldapsearch

Now that you've come this far, you can do a lot of things with your directory. You could, for example, simply take your browser and look for data in your LDAP directory. That isn't the most interesting part though. As an alternative, you can configure your Linux client so that authentication is no longer done on your local password and shadow files, but on the LDAP server, giving you one central point from which to administer all user data instead of hundreds of

computers with all their individual password files. To make it all happen, do the following.

1. Install the Software

Before you can configure your client to authenticate on an LDAP server, you should make sure all necessary software is installed. If you are using RPMs, the packages `openldap`, `auth_ldap` and `nss_ldap` should be present. You can verify that with `rpm -q packagename`. If they are not present, you can find them at rpmfind.com.

2. Edit `ldap.conf`

Often, two files are named `ldif.conf` on systems. One is in `/etc` and is used by `nss_ldap` and `pam_ldap` to determine where they can find required information. The other is in `/etc/openldap` and is used by utilities such as `ldapadd` and `ldapsearch` to determine in which container they should work. As stated before, delete one of them and make a link to the other to make things easy. Once that's done, you can put the necessary data in it. For a simple configuration you only need two lines:

```
BASE    dc=azlan, dc=com
HOST    laetitia.azlan.com
```

The first line specifies the default container where the client should look for data, and the second line gives the name of your LDAP server. Of course, your system must be able to resolve this name by means of DNS or something similar, otherwise you could use an IP address.

3. Edit `nsswitch.conf`

Next, you have to tell the nameservice switch where it should look for data. Do this by editing the file `/etc/nsswitch.conf`; it should contain the following lines:

```
passwd: files ldap
shadow: files ldap
group: files ldap
```

With these lines, your system first tries to authenticate users on your local password files, and if that doesn't work, it tries to authenticate on the LDAP database. So if a user exists in `/etc/passwd`, and he or she gives the password that is in `/etc/shadow`, LDAP will not be used.

4. Edit Your PAM Configuration

Next, you should take a look at PAM. This is the mechanism used on most modern Linux distributions by the different programs that have anything to do with user authentication. It can, for example, be used by `login`, but also by `FTP`,

su, ssh, passwd, etc. In recent versions of PAM, each of these programs has a configuration file, normally in /etc/pam.d. In this configuration file you can specify the PAM modules that should be used by the module.

If you want the login process to do authentication on LDAP, the corresponding configuration file could look like Listing 3.

Listing 3. Login Doing Authentication

Let's give a brief explanation. There are four processes in which user and password information is used. First there is authentication, represented in the PAM file by "auth". This process lets you into the system, and one of its responsibilities is to check your password. Then there is "account", which verifies whether the user has any account restrictions that could prevent him or her from logging in to the system. After that there is "password", which is used if you want to change your password. Lastly, "session" specifies the tasks to be done if you want to use other resources on the system on which you are already authenticated.

Each of these modules has specific tasks. These tasks are specified in the PAM modules, and one of the most important is pam_unix.so. This module takes care of the normal passwd/shadow authentication and is normally required if you want access to the system. But if you are using LDAP, it is also good if LDAP is able to let you in. So before the line where pam_unix is called, there is a line where pam_ldap is called. It is not required (you still want to be able to use your system if the LDAP server is down) but it is sufficient. That is, if you can be authenticated by pam_ldap, you don't have to go to pam_unix afterward. Besides these two major modules, there are some minor modules that are not discussed here.

5. Create the Users with All the Needed Attributes

After you have done the four preceding steps, your computer is ready to authenticate on the LDAP directory. But is your directory also ready? To prepare your directory for authentication, you need to put all relevant user attributes in it, all the information normally in /etc/passwd and /etc/shadow. Otherwise, how could you use your precious resources without a proper user ID and all the other nice things from these files? To get the information, you can use some Perl scripts specially created to get information out of the files on your computer and into the LDAP database, or you can make your own LDIF file to import the users you want.

If you want to do it all automatically, many Perl scripts are available at <http://www.padl.org/>. There are scripts to import almost all settings that can also be in an NIS database—your password files, your host file, your network file, etc.

Before you can use them, however, you have to edit the general configuration file, `migrate_common.ph`. In this file you have to change some parameters that specify the location where the data has to be created. Especially important are `DEFAULT_MAIL_DOMAIN` and `DEFAULT_BASE`; they specify the DNS domain in which users have their e-mail accounts and the LDAP container in which users must be created, respectively. Once that's done, you can start the import. For each different kind of information, there's a separate script; probably the most interesting of them are `migrate_all_online.sh`, which imports all network information, and `migrate_passwd.pl`, which imports the users on your system.

The other way to get things done is by means of an LDIF file, the contents of which must be added to the database using `ldapadd`. The most important thing for this file is that all the right properties are specified. Listing 4 shows an example of how to ensure this.

Listing 4. Sample Property Specification for an LDIF File

There are only two disadvantages to this method. First, the home directory isn't automatically created when you create the user in the LDIF database, but there's a PAM module named `pam_mkhome` so that can take care of it for you. The other problem is user passwords; there's no nice method to get them in encrypted form in the database. A less elegant way to accomplish this is to create the user once in `/etc/passwd` and `/etc/shadow`, give him or her a password and copy the encrypted string out of `/etc/shadow` and into your LDIF file.

Once this is done, you can try the whole thing out. Delete the user from your local files, open a login prompt and try to log in; it should work just fine.



Sander van Vugt (sander.van.vugt@azlan.nl) lives in the Netherlands. He works for Azlan Training as a Linux, Novell and Nortel technical trainer and has written several books and articles about Linux.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

InDepth: Configuring xdm

Ron Hume

Issue #89, September 2001

For both convenience and flexibility, xdm offers a useful way to manage X sessions.

Have you ever wanted to access your workstation's desktop remotely? How about accessing your desktop on a server machine? That's just the sort of problem I needed to solve recently. I was responsible for setting up and managing a group of Linux servers. By the fifth trip to the lab to access a GUI console for various reasons (badge access, key codes, flights of stairs, etc.), it was time to find a solution that would allow me to use my workstation to access X desktops on various server machines.

Some may think that the standard X tools used to handle remote sessions would be sufficient to solve my problem—something like:

```
telnet host1
export DISPLAY=mywkstn:0
gnome-session
```

(or some other session manager).

However, the solution I was looking for needed to provide more than just basic functionality. It needed to be more administrable, appear more automatic and be easier to use for developers new to Linux. There are pitfalls associated with authentication, session management, etc., that require some knowledge of the way X works. For example, I often forget to type **xhost +host1** when using remote X clients. I also have been on the receiving end of a puzzled look when trying to explain the xhost authentication scheme to a Linux newbie. Since this development project was not the time to teach developers the basics of X, I was looking for a solution that fulfilled all of these requirements.

A couple of solutions would allow me to manage the X sessions more easily. The solution I chose for this project was the X display manager, or xdm,

although another popular solution is vnc. I chose xdm over vnc for two reasons. First, vnc has a server-side requirement to have a dæmon running for each shared desktop. Second, I already had X server software installed on all of the workstations and didn't see the need to install additional client software. Other choices are kdm and gdm, part of the KDE and GNOME packages, respectively.

X Basics

X is the graphical support system used in most UNIX environments. If you are using GNOME or KDE on your Linux desktop, then you are making use of the X Window System. It is defined and maintained by the X Consortium (www.X.org). Most Linux users use an implementation of the X Window System offered by the XFree86 Project (www.xfree86.org). **xdm** is a display manager that enables flexible session management functionality. While xdm is usually thought of as “that GUI logon screen that auto-starts my X stuff”, it is actually much more powerful, as we will see.

In the X world, the terms client and server can get a little confusing. Specifically, an X server is the application that controls the keyboard, mouse and display resources. A client is an application that makes requests for the server to perform actions on its behalf (i.e., display a window with some set of specified characteristics). This is a little different for those of us used to thinking of applications running on our workstations as clients.

xdm uses the X Consortium's X display manager control protocol, XDMCP, to communicate with the X servers. This allows X servers to obtain session services from servers running xdm. Three types of queries can be sent by X servers:

- Direct—asks the named host directly to display a login screen.
- Broadcast—broadcasts a message to all hosts on the network, and the first to answer offers the login-processing services.
- Indirect—contacts a named host running xdm and asks it about known hosts with which it may communicate. The xdm server will present a list of available servers willing to manage an X session. The X server will eventually end up communicating directly with the selected host to obtain login-processing services.

One of the initial reasons xdm was created was to allow for management of X terminals. These devices are basically a display, keyboard and mouse with embedded X server software; all intelligence is located on a server in the network. **xdm** was used to push login screens and manage sessions for these devices. Several years ago these devices were popular because access to UNIX workstations was limited. Users who wanted to access graphical desktops at

their desks were either lucky enough to have a UNIX workstation at their desk or required one of these devices. Lately these devices have become less popular and are being replaced by PCs running X server software, such as Linux and other Unices (Solaris x86, xBSD, etc.) or Windows (running Hummingbird Exceed or the like).

When using xdm to manage these X sessions there are some configuration gotchas. At first glance, it may appear that if you configure xdm (in order to take advantage of XDMCP), you get either a local X server started (i.e., the console goes into graphical mode when xdm starts) or, if you disable the local display in xdm and use startx, it doesn't give you access to the chooser. The configuration described here allows any XDMCP client to access the Linux server desktops (subject to X security provisions, of course). It also demonstrates one way to configure xdm in order to get both a local X desktop and access to other server desktops from the workstation.

Security and access control are managed by xdm but are beyond the scope of this article. **xdm** should only be used in controlled environments. In addition, incoming port 177 should be blocked on all firewalls. If you're interested in X security issues, the following man pages are a good place to start: xdm(1), xauth(1), Xsecurity(7), lbxproxy(1)--Low Bandwidth X proxy, xfw(1)--X Firewall Proxy, and ssh(1) and sshd(8) man pages, specifically regarding X11 port forwarding.

Configuring xdm

xdm is highly configurable; the following is only one way to configure it in order to accomplish a specific goal.

On my Red Hat 7 system, xdm lives in /etc/X11/xdm. Its main configuration file is xdm-config (see Listing 1).

Listing 1. xdm-config

xdm's configuration files are in X resource format. There are resources for the configuration of the locations of various files. We are interested in the files pointed to by the resources servers, accessFile and resources. The adventurous will be interested in the session and DisplayManager._X.setup, where X is the display number.

Notice that DisplayManager.requestPort:0 is commented out. This resource specifies which UDP port to use to listen for XDMCP requests. If it is set to 0 (as is the default), then XDMCP requests are ignored, and xdm only manages local displays (see Xservers file). We comment it out so that xdm will listen on the default port (UDP port 177).

My Xservers file looks like this:

```
#:0 local /usr/X11R6/bin/X
```

If this line were not commented out, then I would get a graphical login screen every time I started xdm. That is, it would start and manage a local X server on display 0 by running the command `/usr/X11R6/bin/X`. What we want to be able to do is select the host that we will connect to. We accomplish this using the Xaccess file:

```
#any indirect host can get a chooser
* CHOOSER BROADCAST
#
# If you'd prefer to configure the set of
# hosts each terminal sees,
# then just uncomment these lines
# (and comment out the CHOOSER line above)
# and edit the %hostlist line as appropriate
#
#%hostlist      host-a host-b
#*              CHOOSER %hostlist      #
```

Although the Xaccess file is a very flexible tool, we will only be using it to launch the chooser (indirect mode). The chooser is a little X application that displays a list of available hosts on the network, allowing us to select the one to which we would like to connect. I like to use the BROADCAST option because new hosts show up in the list automatically. Some may prefer to name the hosts specifically, as shown using the %hostlist macro. This method is sometimes required, especially in larger networks where the broadcast doesn't reach all desired hosts.

If you are interested in a finer level of control, you can use a list of servers instead of BROADCAST. This will allow you to specify the list of available hosts directly.

If you want to configure xdm to handle requests from different X servers in different ways, you can specify a hostname or host list instead of *. Examples of this follow.

The following lines tell xdm to handle all queries from either host-a, host-b or host-c itself (Direct mode):

```
host-a
host-b
host-c
```

To tell xdm to send indirect queries from host-a to server-a or server-b, type

```
host-a      server-a server-b
```

It could also be written

```
%hostlist      server-a server-b
host-a         %hostlist
```

You can set up xdm to handle indirect queries using the chooser (our preferred method). In our next example, host-a gets a chooser window containing a list of all hosts that answer the BROADCAST, while everyone else gets only the list specified by %hostlist:

```
%hostlist      server-a server-b
host-a         CHOOSER BROADCAST
*              CHOOSER %hostlist.
```

Finally, to finish up the basic functionality, we can look at the Xresources file. I left mine with the defaults, but some might want to customize the look and feel a bit. In this file you can change colors, fonts and other style options. I've found the Chooser*geometry resource to be the most useful because it allows you to set the size of the chooser application window.

You can configure some administrative functions in xdm-config as well. Things like DisplayManager.errorLogFilelogfile will set the location of the log file. This log file contains the stderr output of xdm, Xsetup, Xstartup, Xsession and Xreset scripts.

Upon successful logon-process completion, xdm launches the script file specified in the session resource. This allows users to customize the behavior of the X sessions. Administrators will most likely want to check out the Xsession script. Users will want to create a \$HOME/.xsession or \$HOME/.Xclients file to customize the behavior of the session manager (i.e., start a window manager, a clock, etc.).

Testing Our Configuration

In order to test our configuration, we need to find X (**which X**). On my system, it's in /usr/X11R6/bin/X. In any case, you should end up seeing a logon screen. To test direct mode you would type

```
/usr/X11R6/bin/X -query remotexdmhost
```

For indirect broadcast mode type

```
/usr/X11R6/bin/X -broadcast
```

And for indirect mode using the chooser, type

```
/usr/X11R6/bin/X -indirect remotexdmhost
```

Once these were working, I created the /etc/rc.d/init.d script to auto-start and auto-stop the xdm service. See the article on using the chkconfig utility in the April 2001 *Linux Journal* for more information.

Then, I created the following scripts to make life simple for my users. On their workstations, I create a file named `/usr/bin/X11/startx.xdmcp`. If the host is called "wkstn1", then the file contains

```
#!/bin/sh
/usr/X11R6/bin/X -indirect wkstn1
```

where hostname is the name of the xdm server (in my case the workstations are both an xdm server and an X server).

Next, I entered

```
mv /usr/bin/X11/startx /usr/bin/X11/startx.original
chmod 755 /usr/bin/X11/startx.xdmcp
ln -s /usr/bin/X11/startx.xdmcp /usr/bin/X11/startx
```

This allows any user who may be used to logging in to their workstation and typing **startx** to get a console, to instead receive a list of available hosts to log in to (including their own workstation).

Conclusion

To recap, we configured the workstations and servers in the network to use xdm, XDMCP in indirect mode and the chooser in order to allow users to select the server they wish to use to manage their X session. This is an administrable solution allowing fine control over X sessions. It is also easy to use, in that it provides users a menu of hosts willing to manage their X session.

If this type of access is required from platforms such as Windows, and you don't want to buy or install an X server on your workstation, then vnc may be an alternative for you.

Resources



Ron Hume (ronhume@ieee.org) is an independent telecommunications consultant. His specialties are in softswitch and enhanced service construction for traditional and emerging telecom service providers.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Password's Progress

Bruce Byfield

Issue #89, September 2001

Some advice on taking advantage of MD5, PAM and more.

If you want an example of evolution in action, look at the GNU/Linux password system. Although it includes the basic UNIX password structure as a vestigial organ, natural selection in the form of crackers has forced the evolution of shadow passwords, the MD5 algorithm and PAM. Passwords have even found new niches in the form of boot managers, remote login formats and advanced security systems. All these tools are used daily, but by taking a closer look at them, you can use them to make your system a little more secure.

Back in the Jurassic days of the 1970s, the standard UNIX password structure appeared. Nowadays, many users access it graphically through gdm, otherwise its structure hasn't changed much. To log in to an account, users enter a password of up to eight characters (it was six or seven in the past), and the password is encrypted into a key using the DES (data encryption standard) algorithm. This key is stored in the second column `/etc/passwd`, where any user can view it.

What has changed is the competition. Nowadays, the DES algorithm can be cracked in seconds. And, to make matters worse, there's little choice in the standard structure except to store the key for each password in a public place, where any intruder can find it. The alternative is to place severe restrictions on ordinary users and prevent them from using basic commands like `ls -l`. Although some security experts would be happy with a computer that was turned off and left in a lead-lined vault several miles beneath the surface of the earth, these restrictions are largely unacceptable.

By the mid-1990s, with the Internet's popularity giving crackers more opportunities, the competition was becoming intense. Because of this pressure, defenses began to evolve. At first available only as add-ons, by the dawn of

modern times in the late nineties, these defenses were in symbiotic relations with each other and were standard parts of every distribution.

The Shadow Knows

Shadow passwords get their name because they are the hidden counterparts of basic passwords. The difference is that, instead of being native to a public file like `/etc/passwd`, their habitat is the second column of `/etc/shadow`, a file readable only by the root user. If no password is set for an account, the column is marked by an asterisk or an exclamation mark, depending on the distribution. Only an "x" in the password column of `/etc/passwd` is left to mark their passage.

Since shadow passwords have been standard, manual configuration has all but disappeared. All the same, shadow passwords generally come with a toolset. **pwconv** and **grpconv** keep user and group entries in `/etc/shadow` and `/etc/passwd` in sync, but this housekeeping is generally done automatically when a password is created for the account. Similarly, **pwunconv** and **grpunconv** allow the creation of regular passwords, but few modern systems ever require this devolution. About the only useful shadow password tool is Debian's **shadowconfig**, whose on-and-off options can tell you quickly whether shadow passwords are enabled.

Another evolutionary dead end is the set of additional columns in `/etc/shadow`. Superficially, these columns promise extra control over when passwords are changed, when warnings of the need to change are given and when an account is disabled if its password is not changed. These columns could be a major survival trait. Unfortunately, they have to be entered individually for each user. More importantly, they must be entered in days since January 1, 1970. This measurement is so cumbersome to calculate that many system administrators leave most of the columns blank and fill the rest with impossibly large numbers so they can ignore them.

The Age of MD5

Another adaptation of the basic password structure is the use of the MD5 encryption algorithm. MD5 is the latest in an evolutionary line of algorithms developed by Ronald Rivest, an MIT professor and a founder of RSA Security, the dominant company in encryption for over a decade. It is a descendant of MD2, an algorithm optimized for 8-bit machines, and a modification of MD4, an algorithm for 32-bit machines that Rivest and his collaborators felt was rushed into release. First released to the public domain in 1991, MD5 was further modified in 1994.

Today, MD5 remains a standard in authentication, even though Rivest insists that it was never intended for that use. More elaborate algorithms have been developed in recent years, such as IDEA, Skipjack or Blowfish, but none has been proven to outperform MD5 consistently enough to replace it.

MD5 became available as an add-on for GNU/Linux in the mid-nineties and is now a standard part of most distributions. From a security perspective, the advantages of MD5 over the DES used in the standard password structure is that it allows for longer passwords and provides more sophisticated encryption. When MD5 is enabled, passwords of up to 256 characters are possible. Regardless of the password's actual length, MD5 passes it through four rounds of encryption to create a 256-character key. Since this process is not reversible (at least, not without considerable effort), MD5 is classified as a "one-way hash".

MD5 is an option during the installation of every major distribution. Although MD5 can create problems with network information systems on most modern workstations or networks, there is no reason not to use it. If you are unsure whether MD5 is enabled, check whether the password column in `/etc/shadow` starts with `1`, or search the files in `/etc/pam.d` for lines that end in "md5". If it isn't, locating the necessary files and reconfiguring the system is time-consuming enough that a new user might be tempted to upgrade or re-install instead.

Articulating the Skeleton: PAM

The rise of shadow passwords and MD5 could potentially cause over-diversification, with every combination of add-ons requiring its own versions of commands like `passwd` or `login`. This problem is avoided by the Pluggable Authentication Method (PAM). PAM can be thought of as an intermediary between the commands and processes involved with authentication and any modifications to it. PAM evolved along with shadow passwords and MD5 and has been available in distributions since about 1997.

Originally, PAM was configured in `/etc/pam.conf`. However, in most distributions, this file is now as vestigial as an appendix. Instead, PAM uses the `/etc/pam.d` directory.

A handful of files in `/etc/pam.d` define which users or groups, if any, can use a specific command. For example, `/etc/pam.d/su` regulates the `su` command. Other limits may be set in `/etc/security/limits.conf`. However, the majority of the files in `/etc/pam.d` act as intermediaries between password system enhancements and other commands, pointing to libraries in `/lib/security`. Examples of these files include `chfn`, `chsh`, `cron`, `gdm` and `login`. This function

allows not only the use of shadow passwords or MD5 but eases the addition of upscale security solutions such as Kerberos.

The files in `/etc/pam.d` offer far too many options to detail here. However, the files are heavily commented and relatively easy to follow. Files such as `passwd`, `gdm`, `login` and `su`, which control the basics of the password system, are especially useful. For example, `login` can control root logins, put a time restraint on logins and set how login attempts are logged. Similarly, if you use `su` on the system, rather than the more controllable `sudo`, `/etc/pam.d/su` can help you set limits on how the command is used. And, while you don't want to change the references to security libraries in the `pam.d` files, you might want to look at the options used with them—for instance, the security-minded might want to avoid `nullok`, which allows users to change empty passwords. Going farther afield, `chsh` can be used to limit the shells users can use to a list in `/etc/shells`. In short, while browsing the `pam.d` directory can leave you with the trauma of option anxiety, you'll find the effort an important step in learning how to make your system more secure.

Evolutionary Pressures and New Niches

Shadow passwords, MD5 and PAM all increase the security of a system. However, keep in mind that the security they provide is relative. Given enough computing power and enough time, a brute-force attack can crack any system.

Moreover, the effort is getting easier with each advance in hardware and crackers' tools. To put things in perspective, in 1994, RSA Security system estimated that a brute-force attack on the average machine would succeed within 24 days. By contrast, the developers of `mdcrack`, a tool that can be used for testing the security of MD5-enabled systems, claim that a 56-character password can be cracked on an average machine using the 2.2 Linux kernel within 20 seconds. Although that was almost twice as long as the average time to crack a Windows machine, obviously Linux users have no reason to be smug. And the situation is only going to get worse.

One way to respond to this pressure is to make better use of the password system. Many users, especially at home, forget about the password system after they install and fail to use anything except its most basic features. Yet a little attention to detail could be enough to send the script kiddies into tantrums. For example:

- Set the number of days that a password can be used in `/etc/shadow`. The method is a pain, but regularly changed passwords could set back a brute-force attack that relies more on time than computing power.

- Increase the minimum and maximum password lengths in `/etc/pam.d/passwd`. Other things being equal: the longer the password, the longer it takes to crack.
- Lower the minimum number of login attempts in `/etc/pam.d/gdm`. Legitimate users with sausage-like fingers might complain, but anyone attempting a brute-force attack could be irritated enough to go away.
- Approve all user passwords or insist that they are generated by a program like `pwgen` that creates secure passwords. The number of users whose password is “password” or the name of their youngest daughter or goldfish is too depressing for words.
- Install a program like `cracklib2`, which prevents the use of easily guessed passwords. You'll probably want to add a custom dictionary of company, product and user names that cannot be used as passwords. Currently, `cracklib2` requires some patience to set up, as well as the uncommenting of a number of lines in `/etc/pam.d`, but it, or a similar package, is likely to become a standard part of major distributions in the next few years.
- Do not allow the root user to log in remotely.
- Watch for expired passwords or user accounts without passwords. Either could be an entry point to the system.
- Turn off your computer or your net connection if you're not using it. Forget the geek macho about how long your system has been running. If it's not connected, then remote cracks can't happen.

Another response to the increased risks of modern computers is the increased use of password authentication on the system. Here are some of the places you can add passwords if they aren't already there:

- The BIOS: use a password and make sure it can't be bypassed via floppy or CD.
- The Boot Manager: use LILO's `password` command or GRUB's `lock` command.
- Remote Services: `ssh` sends encrypted passwords—Telnet and FTP don't. Guess which one you should use?

Of course, passwords alone won't secure your system from anyone except the rawest of script kiddies. And, the harder the password is to crack, the harder it is to remember—and the more likely that the user will write it on a post-it note taped to the bottom of the keyboard.

Still, there's no reason not to use whatever security passwords afford. And there's definitely not a reason to bypass the password system or weaken it—two options that are starting to appear in modern distributions in the hopes of

making GNU/Linux seem more like other operating systems of the average user's acquaintance. The tools are there, so why not use them?



Bruce Byfield is a contract technical writer, product manager and journalist. Away from his computer, he consorts with exotic birds, listens to punk-folk and runs long, painful distances for pleasure. He can be reached at bbyfield@axionet.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Loadable Kernel Module Exploits

William C. Benton

Issue #89, September 2001

Beat potential invaders at their own game by learning how to use cracker tools and improve your own security.

Many useful computer security tool ideas have a common genesis: the cracker world. Tools, like port scanners and password crackers, originally designed to aid black-hats in their attempts to compromise systems, have been profitably applied by systems administrators to audit the security of their own servers and user accounts. This article presents a cracker idea—the kernel module exploit—and shows how you can improve your system's security by using some of the same ideas and techniques. First, I will discuss the origin of my idea and how it works, then I will attempt to demystify the art of kernel module programming with a few short examples. Finally, we will walk through a substantial, useful example that will help prevent a class of attacks from compromising your system.

Before we get started, I need to mention the standard disclaimer. Be aware that a bug in kernel space is liable to crash your machine, and an endless loop in kernel space will hang your machine. Do not develop and test new modules on a production machine, and test modules thoroughly to ensure they do not destabilize your system or corrupt your data. To minimize data loss due to system crashes in the debugging cycle, I recommend that you either use a virtual machine or emulator (like bochs, plex86, the User-Mode Linux port or VMware) for testing, or install a journaling filesystem (like SGI's xfs) on your development workstation. Furthermore, none of the code examples in this article have been tested on an SMP machine, and most of it is likely not multiprocessor safe. Now that we have that out of the way, let's talk about modules.

A few months ago, I was developing a system called audit trail generator for Linux. For every process on a system, I wanted to keep track of all system calls and their arguments. To this end, I experimented with several approaches, but

none was as successful as I would have liked. Wrapping the libc function for write(), for example, only enabled me to log write() invocations that originated from C programs, and dynamic binary instrumentation was limited by the sorts of executables the instrumentation library could parse (C, C++ and Fortran). Being limited to auditing executables produced by one of a few languages was only a small practical limitation, since virtually every program on a GNU/Linux system is written in C, C++ or some language that has a C- or C++-based runtime library, like Perl or Python. However, the incompleteness of these solutions really bothered me on a theoretical level. I knew how straightforward it would be to bypass this system by invoking a system call from a little-known language that didn't rely on C or C++, or even by handcrafting a system call in assembly language. It was clear that it would be impossible to write an insubversible user-space auditing tool, and it would be tough to write a really useful tool without hacking into the kernel. Since I didn't want to maintain a patch or deal with a lengthy recompile-reboot-debug cycle, I didn't think doing this in kernel space was feasible.

No sooner had I put these concerns on the back burner and started work on this project than I saw a message to my local LUG's mailing list that gave me an idea. This message was a forwarded advisory about a kernel module exploit. This particular module was a nasty one: it modified the behavior of certain system calls to hide itself from the lsmod command and to hide the presence of scanners, crackers, sniffer logs and other such files. I almost screamed "Eureka!" in my office. I didn't have to deal with maintaining a kernel patch, recompiling or rebooting; I could develop my tool as a loadable module. I recognized that the general technique behind module exploits could be adapted to add many types of useful behavior to system calls, including a different security policy, finer-grained security than the UNIX model allows and, of course, my audit trail generator.

Hello, Kernel!

I will discuss some of the fun things you can do by altering and wrapping system calls a little later, but let us first get our hands dirty with an example kernel module. This is a simple example, akin to everyone's favorite first program, but it demonstrates the most basic parts of a loadable kernel module, the init_module and cleanup_module functions:

```
#include <linux/kernel.h>
#include <linux/module.h>
int init_module() {
    printk("<1> Hello, kernel!\n");
    return 0;
}
void cleanup_module() {
    printk("<1>I'm not offended that you"
        "unloaded me.  Have a pleasant day!\n");
}
```


You may have to use `#define` for the symbol `MODVERSIONS` and `#include` for the file `linux/modversions.h` from the Linux source tree, depending on how your system is set up. Call this short module `hello.c` and compile it with:

```
gcc -c -DMODULE -D__KERNEL__ hello.c
```

You should now have a file called `hello.o` in your current directory. If you're currently in `X`, switch over to a virtual console and (as root) type **`insmod hello.o`**. You should see “Hello, kernel!” on your screen. If you would like to check that your module is loaded, use the `lsmod` command; it should show that your `hello` module is loaded and taking up memory. You can now **`rmmod`** this module; it will politely inform you that you have unloaded it.

The `linux/kernel.h` and `linux/module.h` header files are the two most basic for any module development, and you are likely to need them for any module you write. It is best if these headers (unlike `modversions.h`) come from `/usr/include/linux` rather than a Linux source tree. (If your distribution vendor has made `/usr/include/linux` a link to the Linux source tree, complain—that practice is liable to cause major breakage and headaches for you.) You will use quite a few more of the kernel headers for any substantial module, and you will find that

```
grep -l /usr/include/linux
```

is a good friend while developing modules.

Think of `init_module` as an “object constructor” for your module. **`init_module`** should allocate storage, initialize data and alter the kernel state so that your module can do its work. In this case, `init_module` is merely announcing its presence and returning 0 to signify success, as in many C functions. Therefore, our initialization for the `hello` module consists solely of calling the `printk` function, a particularly handy function to have at your disposal. Essentially, it functions like the standard C `printf` function, but for two differences. First, and most obviously, `printk` allows you to specify a priority for a given message (the “1” in angle brackets). Second, `printk` sends its output to a circular buffer that is consumed by the kernel logger and (possibly) sent to `syslogd`. Since the output of `syslog` is flushed frequently, calling `printk` with judiciously placed, high-priority messages can greatly aid debugging—especially since any bug in kernel-space code is liable to crash your machine or at least cause a “kernel oops”.

Why not just use `printf`, you ask? Simple: to do so would be impossible. The Linux kernel is not linked to the C library, so old friends like `printf` are unavailable in kernel-space code. However, there are many useful routines in the kernel that give you functionality similar to library routines, including workalikes for most of the `str` family of functions from the C library. To use

these in your modules, merely include `linux/string.h` (be careful not to include the C library version).

If `init_module` is a constructor, `remove_module` is the destructor. Be sure to tidy up after your module as carefully as possible; if you don't free some memory or restore a data structure, you'll have to reboot to return your system to normal.

A More Interesting Module

Now we graduate to a more advanced example. Listing 1 presents a module that logs a message every time someone other than uid 0 (root) or uid 500 (me on my workstation) invokes the write system call with the word "Linux" somewhere in the buffer. You may have to stretch a little to find a use for this module by itself, but I assure you it demonstrates several useful concepts. We are able to do this all by replacing the write system call with our own function that performs the checking and logging, and then calls write. Let's go through this example step by step.

Listing 1. Checking and Logging Function

Notice all of the include files. There sure are a lot of them, but don't despair, the ones we are going to worry about are `linux/sched.h` and `asm/uaccess.h`. The `sched.h` include allows you to access the current `task_struct` structure via the `current` macro, providing a great deal of useful information about the current process (see Table 1 for a list of some useful fields in `task_struct`), while `uaccess.h` provides useful macros for accessing user-space memory (more on this later).

Table 1. Useful task_struct Fields

Even these few fields in `task_struct` are enough to enable some really interesting modules. Should arbitrary users be allowed to `su` to root? You can prevent them from doing so by wrapping `setuid` and checking for one of several prespecified UIDs before allowing the "real" `setuid`. This will allow you to develop, at the kernel level, an equivalent to the wheel group, or group of users that are allowed to `su` root. As an aside, the FSF has long held that the wheel group is a tool of fascist administrators (see the documentation for GNU `su` for more information).

Being able to audit or alter the behavior of system calls, simply on the basis of which uid invokes them, is obviously a powerful ability. It can make for good security policy to control and audit the actions of the "nobody" user and its friends, the `uucp`, `mail` and `postgres` users carefully. However, an even more powerful technique is to alter behavior based on an argument. We will ignore

`sys_call_table` and `origwrite` for now and proceed directly to `wrapped_write`, which examines both the `uid` of the invoking process and its `buffer` argument.

The first thing you should notice is that `wrapped_write` begins with a call to `kmalloc`. Why not `malloc`, you may ask? Remember, we're still in kernel space, and we don't have access to `malloc` and other standard library functions. Even if we did, calling `malloc`, which returns a pointer to user-space memory, would be worthless. We need to allocate some memory in kernel space to copy data into from the `buf` argument. This is an important point: the same memory visibility barrier between kernel and user space that keeps your programs from crashing the kernel also adds a little bit of complexity to your kernel programming. When you call `write` from a C program, you pass a pointer to a user-space memory block that is inaccessible from the kernel. Therefore, if you want to do any operations on data pointed to by a user-space pointer, you will have to first copy that memory area into kernel space. The `copy_from_user` macro does this for you. **`copy_from_user`** takes three arguments, a "to" pointer, a "from" pointer and a count.

The remainder of `wrapped_write` is fairly straightforward, given what we know about `current` and `task_struct`. Perhaps a more interesting module would use `strstr` to check for the string "Linux sucks", and if it existed, alter `write_buf` at that point to contain "Linux rule", then transfer `write_buf` back to user space (with the `copy_to_user` macro) before calling the original `write`. Then, if unsuspecting users wrote "Linux sucks", it would be replaced with "Linux rules". **`kfree`** is important here. Leaking memory in the kernel is a bad thing, so be sure to **`kfree`** everything you **`kmalloc`**.

It is in `init_module` that we actually make the switch so that our function is called instead of the original `write`. Recall that `sys_call_table` is an array of pointers to functions. By altering the value at index `SYS_write` (a constant representing the system call number for `write`), we are able to cause another function to replace `write`. Be sure to save the original function, so you can replace it when the module is unloaded! You can test this module out by compiling and installing it with `insmod`; then `su` to some user other than 0 or 500, and type

```
% echo "I like Linux"
```

on a virtual console. You should get a message from the kernel that you're talking about Linux again. Congratulations! You are now ready for a module that does something useful.

A Final Example

Listing 2 [available at <ftp://linuxjournal.com/pub/lj/listings/issue89/4829.tgz>] demonstrates a useful module that can help prevent your system from falling victim to stack-smashing attacks. A stack-smashing attack basically consists of writing past the end of a fixed-size buffer, so that the return address of the current function is overwritten, usually with a jump to `exec (/bin/sh, ...)`. Since there really is no reason for programs like `httpd`, `fingerd` or `wu-ftpd` to **exec** a shell, we shall provide a mechanism to disallow it. By this point, you already have the knowledge to understand most of the code, with one small exception: the `strncpy_from_user` function. As you might expect, it functions much like its C-library counterpart, `strncpy`, and is a handy way to get a null-terminated string from user space. Since the code is straightforward, we'll briefly discuss the approach, and then I'll leave you to come up with great ideas of your own for improving your system's security.

The implementation in Listing 2 is straightforward. It is not as efficient or robust as one might want, but this code was written in the interest of clarity, and it is easy work to make it better by changing the linear search in `wrapped_execve` to something more efficient. Essentially, what this module does is overload the kill system call so that if you send signal 42 to a process; it is added to a list of "unsafe" processes, processes that should not be allowed to execute any binary with "sh" in its filename. (42 is one of the real-time signals; you probably aren't using it. If you are, feel free to substitute any number between 32 and 64.) The `execve` system call then checks to see whether the process is an unsafe one and, if so, checks to see if it is trying to execute a shell. If so, it returns success without doing anything. It is easy to use this module for all of your server processes; simply add this to your init scripts:

```
kill -42 ...
```

Listing 2 represents an evolutionary step from Listing 1, but it shows that one can modify the behavior of calls, not just add behavior to the call path. It also does useful work. I hope that you are as excited as I am about the possibilities of writing kernel module exploits to improve your security. This article has given you the basic tools to get started. Fortunately, there is a wealth of documentation available to Linux programmers that will help you write more complex and functional modules; see the Resources section.

Resources



William C. Benton (willb@acm.org) is a graduate student at the University of Wisconsin. He is interested in performance monitoring of parallel programs and in language and compiler approaches to security problems.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Introducing Enhydra

Reuven Lerner

Issue #89, September 2001

How to write servlets and basic applications using the tools that come with Enhydra.

Last month, we looked at XMLC, a new system for displaying dynamic web content that comes with the Enhydra application server. This month, we will look at how to write servlets and basic applications using the tools that come with Enhydra. While Enhydra's web applications are not as standardized as servlets with Jakarta-Tomcat, they do offer a fair amount of power and the possibility of creating Enterprise JavaBeans with a fully open-source infrastructure.

History and Background

Enhydra is an open-source application server written in Java, aiming for full compliance with Sun's J2EE specifications. Lutris, the company that spearheads Enhydra development, has made the application server available under a BSD-like license. The current stable version of Enhydra is 3.x and includes support for a large number of standards, including servlets and JSPs. Enhydra Enterprise, which is slated for release during the summer of 2001, will have additional J2EE features, including support for Enterprise Java Beans (EJB).

Enhydra itself is available as a fully open-source product, meaning that you can download it from the Web and install it. But for those clients who might be suspicious of open-source software, who want to use a packaged product that has gone through quality assurance or who want to get support from Lutris, commercial versions of Enhydra are available. I suspect that most people reading this column will not need support from Lutris, but it is good to know that they are ready and willing to help others use the product.

Lutris is aiming Enhydra not just at the J2EE market but also at the market for wireless internet applications. I personally have yet to see a compelling use of

cellular internet technologies—the WAP functionality on my mobile phone could charitably be called pathetic—but Lutris is positioning Enhydra as a player in what will inevitably become a hot market.

While Enhydra has yet to develop the mindshare or community of Zope and the ArsDigita Community System, they have managed to score a fair number of impressive commercial triumphs. In particular, Hewlett-Packard recently announced that they would work with Lutris to market and use Enhydra in a number of applications. If nothing else, this proves that open-source application servers do have an important place in the world of web application development and that they can provide a compelling case even to companies that could otherwise afford to pay much more.

Getting Started

Now that we've reviewed a bit of the background, let's try to create some basic web applications using Enhydra. The first task, of course, is to download and install the product. I decided to download the beta of Enhydra Enterprise, in no small part because I wanted to play with the EJB features. As of the beta release, Enhydra Enterprise requires JDK 1.3, a pleasant contrast to previous releases that required JDK 1.2 even after 1.3 had been released for quite a while.

I downloaded and unpacked the Enhydra tar file, which created a large number of files and directories under the enhydra4.0 directory. As you might guess, the doc directory contains documentation, the lib directory contains .jar files that Enhydra needs and the conf directory contains global Enhydra configuration files. There is also a bin directory, in which nearly all files are shell scripts (with Windows .bat equivalents) that execute the various Java programs that make up Enhydra.

To ensure that the Enhydra shell scripts and Java programs are aware of the directory in which you installed Enhydra, run the configure shell script in the root directory of the Enhydra distribution (to which I will refer as \$ENHYDRA). **configure** takes a single mandatory argument, the root directory of your JDK installation. **configure** modifies a number of Makefiles and other configuration files but does not force a recompilation of any code. After running configure (which does not produce any visible output), run the bash shell script (setup.bash) in \$ENHYDRA, which adds the JDK executable directory to your PATH.

Enhydra consists of a number of different interrelated software packages. The application server itself, known as the multiserver, can work directly opposite HTTP clients or with a front-end web server such as Apache acting as a proxy. You can reduce the number of services that the multiserver starts up or change

the order in which they are started by changing the `loadOrder` property in `$ENHYDRA/conf/bootstrap.conf`.

To start the multiserver, simply run `$ENHYDRA/bin/multiserver`. It should start up right away, launching services one at a time until you finally see the message, "Bootstrapper initialized normally". At this point, you can test the multiserver by pointing your web browser at port 8001, which should bring up a control panel for viewing the current state of your multiserver.

My initial attempts to launch the multiserver failed, with the program complaining that it was unable to find `enhydra.jar` in my `CLASSPATH`. Particularly confusing here was the fact that I couldn't find any such file as `enhydra.jar` in the entire Enhydra distribution.

The solution turned out to be simple, if not obvious: Enhydra knows what `CLASSPATH` is necessary in order to run each of its programs but will ignore those settings if you have already set your `CLASSPATH`. So before running the multiserver, make sure to unset `CLASSPATH`, removing this environment variable. Once you have done this, the multiserver should come up as expected.

Building a Simple Web Application

Over the last few months, we have looked at Java servlets and JavaServer Pages. Enhydra, as a J2EE-compliant application server, fully supports these technologies. Moreover, as an open-source server, Enhydra uses the Jakarta-Tomcat engine as the basis for its servlets and JSPs. As we will see later, it is possible (and often preferable) to use Enhydra's own advanced form of web applications.

It's relatively easy to create servlets using the tools that come with Enhydra. And indeed, Enhydra's authors have spent a great deal of time creating a system that is not only powerful when deployed, but relatively easy to work with during development.

If you are used to simply writing a servlet, compiling it and dropping it into a directory, then you will find that Enhydra gets in the way, complicating the process. This is in no small part due to the way in which Enhydra applications are deployed—rather than requiring an external server, Enhydra expects that you will want to test (and run) many of your web applications independently of any others.

To create a simple servlet, we will use the application generation wizard (`appwizard`) that comes with Enhydra, which you can invoke as `$ENHYDRA/bin/`

appwizard. The appwizard is not an IDE but rather a sophisticated file-copying program that provides a basic skeleton application that already works.

When you first run appwizard, it will ask you whether you want to develop a web application (i.e., standard servlet) or an Enhydra super servlet. Choose a standard web application; super servlets will come later. The next screen will ask whether you want to produce output in HTML or WML, the latter being the standard XML-based format for cellular internet applications. We will use HTML and will call both the project directory and the package "atf". By default, Enhydra applications are placed under your home directory in the subdirectory enhydraApps. Choose a license under which your code will be released, and appwizard will generate files for your new application.

And indeed, appwizard creates a large number of automatically generated files and directories. Among them are:

- a global Makefile that allows us to build the application. There are individual Makefiles in a number of the applications subdirectories as well.
- config.mk, which defines a number of environment variables on which the Makefile depends, with such information as the Enhydra version, the location of your JDK installation and the location of your Enhydra installation.
- the src directory that contains the source code for Java servlets and HTML files. Under src is a standard WEB-INF directory, whose web.xml file names each of the servlets we plan to deploy. The atf directory, whose name depends on the project we created, contains four subdirectories: business, data, presentation and resources. The two that interest us most are the presentation and resources directories, since the former contains servlets and the latter contains HTML files and JSPs.

To build the application, simply run a **make** in the root directory of our project. (The Enhydra Enterprise documentation makes a big deal out of saying that it now uses the Java-based Ant build tool in favor of make, but application creation still appears to rely on make.)

After make completes its work, there will be a new output subdirectory at the top level of our application, parallel to src and input. The output directory contains everything we need in order to launch our application, including a standard Java .war (web archive) file containing our .class files, XML descriptors, JSPs and images:

```
WEB-INF/classes/atf/presentation/WelcomeHTML.class
WEB-INF/classes/atf/presentation/WelcomeServlet.class
WEB-INF/classes/atf/presentation/RedirectServlet.class
media/Enhydra.gif
```

```
index.jsp
WEB-INF/web.xml
```

Notice that we have three .class files here, while there were only two in src/atf/presentation/. That's because XMLC turned the HTML file from src/resources into a Java source file, which was then turned into a Java .class file.

So with just two commands, appwizard and make, we have managed to create a full, running Enhydra application. The application, as it stands, doesn't do anything particularly complex or interesting, but it provides us with a skeleton that we can then modify and extend.

To run our application, we run **output/start4**. This starts the application on port 9000 (defined in input/conf/servlet/servlet.conf.in). If you point your web browser to http://localhost:9000/, you will see the output from our servlet: the Enhydra logo, the name of our application (atf), the current time and date and a hyperlink that redirects you back to the application.

The HTML page is generated by XMLC and demonstrates how XMLC is integrated into the rest of Enhydra. XMLC compiles src/atf/resources/Welcome.html into a Java servlet, which is then turned into a .class file. The Java class created by XMLC includes a hook for each of the tags in the file, allowing retrieval and modifications of anything within a tag that has an ID attribute.

WelcomeServlet, the servlet that is initially executed in our application, displays the current time and date by creating an instance of the XMLC-generated class:

```
now = DateFormat.getTimeInstance(DateFormat.MEDIUM) .format(new Date());
welcome = new WelcomeHTML();
welcome.getElementTime().getFirstChild() .setNodeValue(now);
```

In other words, we replace the text within the tag with an ID of "time" by turning the HTML file into a DOM-accessible tree and then changing the value of a specific node.

To add additional servlets to our application, we can write and save them in src/atf/presentation. Note that the package will be atf.presentation and not simply atf. We will write a very simple class, Foo, which you can see in Listing 1. Except for the package name, there isn't any difference between traditional servlets and Foo.java.

Listing 1. Foo.java

Now we need to tell the servlet engine to map a URL to our servlet. We do this in src/WEB-INF/web.xml. This XML file is divided into two parts: the first maps servlet classes to servlet names, and the second maps servlet names to URLs.

Listing 2 contains a version of web.xml modified to handle mapping our Foo servlet.

Listing 2. Modified Version of web.xml

Finally, we need to include our new class in the Makefile, adding the name of our class to the CLASSES variable:

```
CLASSES = WelcomeServlet \  
RedirectServlet Foo
```

Run **make** and check that Foo.class has been added to the application with:

```
jar tvf output/archive/atf.war
```

If it all works, then run **output/start4**, and point your browser to `http://localhost:9000/foo`. You should see the HTML output delivered by our new Foo servlet.

Building an Enhydra Application

Experienced web developers, regardless of the language or environment, are used to writing a separate program for each web page. If you want to display five different dynamically generated pages, then you must write five different CGI programs, mod_perl handlers, servlets or JSP pages.

Enhydra allows developers to break away from this model, thinking in terms of applications rather than individual pages. The way to do this is with super servlets, as they are known, in which a single application object is associated with multiple presentation objects.

You can easily identify a presentation object in an Enhydra URL; the suffix `.po` tells Enhydra that it should invoke the object named in the URL. So requesting `Abc.po` will execute the `run()` method for the presentation object `Abc`. Unlike standard Java servlets, presentation objects are instantiated once for each HTTP request. This may be less efficient than using multiple threads on a single-servlet instance, but it does remove the headaches associated with writing threadsafe servlet code.

A simple Enhydra application will thus consist of at least one application object, plus at least one presentation object. These POs, as they are known, can then connect to the two other main types of Enhydra objects: business objects (which perform commonly needed functions) and data objects (which map persistent storage, such as a relational database, to a Java class). Each of these three types of objects—presentation, business and data—has its own directory within an application's `src` subdirectory, as we have already seen. Moreover,

each of these objects constitutes one of the three standard tiers in a three-tier web application. So while it might take some time to get used to the separation between object types, this model is becoming increasingly prevalent in web applications.

Once again, we will use Enhydra's `appwizard` to create a skeleton application that we can change. Run `appwizard` again, but choose super servlet from the selection list on the first screen, rather than a simple web application. I chose to call the project `myproject` and to put it in the `il.co.lerner` package, which is what I use for internal projects at my company. `appwizard` then creates a skeleton application in `~/enhydraApps/myproject`. The application has a similar structure to our servlet, with a similar directory structure. Under `src/il/co/lerner`, we have `presentation`, `data` and `business` directories. And once again, there is a top-level `Makefile` that will compile and create our super servlet.

Look at `presentation/WelcomePresentation.java`, the source code for the presentation object that will eventually be displayed. Indeed, if we type `make` at the top-level directory, run `output/start4` to start our application and point our web browser to `http://localhost:9000/`, we will find that our browser is redirected to `http://localhost:9000/WelcomePresentation.po`. This page displays the same sample output that our skeleton servlet printed, with the Enhydra logo and the current time and date.

The `po` suffix, as we already know, tells Enhydra to invoke the `run()` method in `WelcomePresentation`. In the automatically created skeleton application, `WelcomePresentation.run()` looks like Listing 3.

Listing 3. WelcomePresentation.run()

The super servlet interface is similar to that of a regular servlet and does not take much time for a programmer familiar with servlets to learn. The `run()` method takes a single argument of type `HttpPresentationComms`, which provides our presentation object with all of its communication needs to the outside world, including the HTTP request and response objects.

The `run()` method displays output by creating an instance of `WelcomeHTML`, the Java class that XMLC created from the file `Welcome.HTML`. Following that, `run()` replaces the contents of the `` tags with an ID of "time" with the current date and time. Then we write the contents of `welcome`, which contains a DOM tree, to the HTTP response object.

We can create our own presentation object, `FooPresentation`, as demonstrated in Listing 4. Remember to add the new object to the `CLASSES` line in the presentation directory's `Makefile`. When you rerun `make` from the top-level

application directory, FooPresentation will be compiled and inserted into our Enhydra application.

Listing 4. FooPresentation.java

It's very nice to be able to write our own presentation objects, but where is the application object that is controlling them? In the main source directory, at the same level as the presentation, data and business directories, there is a Java class file whose name is the same as the project—so in our case, there is a file at `src/il/co/lerner/myproject.java`.

Deploying Our Super Servlet

Until now, we have only run our super servlet on its own, outside of the overall multiserver. This functionality is great for developers who want to be able to test applications without interfering with the main production web site, but we will want to add our application to that server at some point.

Enhydra makes this relatively easy to do: we add information about our application to the multiserver's configuration file, so that it can find the application. Then we use the multiserver's control panel to add our application to the production server under the URL of our choice. At that point, our application is available for all the world to see.

In order to accomplish this, we must start the multiserver once again, with the shell script `$ENHYDRA/bin/multiserver`. This makes the multiserver available on port 8001. Load the administration screen in your browser, and look at the list of available applications in the top-left corner.

Now we will copy the application configuration file—but not the application itself—named `output/conf/myproject.conf` and copy it to the global multiserver directory, `$ENHYDRA/apps`. Then edit `$ENHYDRA/apps/myproject.conf`, changing the value of `Server.ClassPath[]`. There are already two possible values for `Server.ClassPath[]` in `myproject.conf`: one for running the application in standalone mode and another for running it under the multiserver. Comment out the (first) standalone value, and uncomment the (second) multiserver value.

Having done this, return to your web browser and click on the Add button (with a big + sign on it) in the multiserver control panel. We're going to add a new application, whose name (`myproject`) should be in the selection list. Choose a root URL for this application, and enter any text string you want for this application's group. Click on OK to add the application.

Now refresh the multiserver control panel. In the upper left-hand corner, you should see “`myproject`”, along with whatever other applications might currently

be loaded. If you click on the myproject name, the right-hand side of the screen will fill with information about myproject.

To run the application, we need to define one or more connections for our application and then run it. By default, our application will run on ports 8002 and 8003; we can add one or more new connections if we want. Once the connections are defined, click on the Run button on the left side of the screen. The connection URLs will become hyperlinks, and clicking on one of them will open a connection to our web application—with the Enhydra logo and current time—displayed in a new window. (I normally find JavaScript and new windows to be annoying, but the Enhydra authors have somehow managed to balance taste and functionality with a decent-looking web interface.)

We can test our FooPresentation object by changing the URL, replacing WelcomePresentation.po to FooPresentation.po. Sure enough, we see our simple Foo HTML output displayed in the browser.

We can remove our application from one or more ports or from the multiserver entirely using the web-based control panel. Finally, we can shut down the multiserver either using the control panel or by pressing Ctrl-C in the terminal window where we started it.

Conclusion

It is not inherently difficult to write servlets, but Enhydra offers much more than just servlets. In particular, they provide an environment that makes it easy to write and test servlets without having to run a full web server. Moreover, the super servlets that Enhydra provides can be easier to work with than regular servlets, especially since we can avoid having to deal with threading issues and writing a new overall handler application for each page.

There are, of course, some drawbacks. Enhydra, like most Java programs, requires a fair amount of patience when setting CLASSPATH. (Although to the credit of Lutris, removing my own CLASSPATH solved almost all of those problems.) And while Enhydra's automatically generated Makefiles dramatically reduce the amount of thought that you must put into creating a finished web application, Java programs always seem to require ten times as many files as their Perl and Python counterparts.

While super servlets are certainly an improvement over their nonsuper cousins, I always hesitate before jumping into a technology that deviates from a known, well-documented and mature standard—particularly when the Open Source community seems to be taking its time to rally around Enhydra in a major way. Finally, while it's true that Enhydra Enterprise is not yet a released product, the installation instructions and documentation leave something to be desired.

With all of these reservations, I easily can see myself using Enhydra for future Java development, instead of the plain, vanilla Jakarta-Tomcat that I have used in the past. The combination of XMLC and an integrated environment is quite attractive in many ways.

As I mentioned earlier, one of the reasons I am most excited about Enhydra Enterprise is its ability to connect to Sun's Enterprise JavaBeans. Over the next two months, we will look more closely at Enhydra, first investigating its DODS tool for mapping relational databases to Java objects. Then we will dip our toes into the world of EJB, demonstrating that just because we rely on open-source products doesn't mean that our tools are any less capable than proprietary programmers.

Resources

Reuven M. Lerner owns a small consulting firm specializing in web and internet technologies. He lives with his wife Shira and daughter Atara Margalit in Modi'in, Israel. You can reach him at reuven@lerner.co.il or on the ATF home page, www.lerner.co.il/atf.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Brochettes de Sécurité

Marcel Gagné

Issue #89, September 2001

Marcel demonstrates different ways to recover lost files and hide and back up secret ones.

Ah, François. Come here and take a look at this. The theme of this issue is security, and I have developed a menu around that very theme. *Quoi?* But of course it is not about networks. Do you know, *mon ami*, what the most important part of running a successful restaurant is? *Bien sûr*, a well-stocked wine cellar is important—I agree with you, but I was thinking about the food, François. You see, our guests, they hunger for many things, and it is our duty as restaurateurs to make sure that their taste buds are constantly stimulated in ways they do not expect.

Qu'est-ce que tu dis? François, security is a many-faceted aspect of cooking with Linux. Network security is but one such item, and everyone no doubt will be thinking network security. By selecting different ingredients, we keep the palate alert and refreshed. For instance, look at this first item. You cannot deny its security implications and yet—François, why are you not paying attention? Eh? Ah, *mais pardon*, I see our guests have arrived. Please, everyone, sit down. I shall have François fetch some wine *immédiatement*.

François! We have a 1998 Clos de Vougeot in the cellar that I think will satisfy everyone's taste buds tonight and add a special air of *je ne sais quoi*. A lovely Burgundy, *mes amis*. *Vite*, François. *Vite!*

François and I were discussing different ways to look at the concept of security. When we cook with Linux, we have all gotten used to thinking of “security” as network security but, as a visit to Monsieur Roget and his famous thesaurus will tell you, security can mean many other things. You will find words like bond, debenture, deposit, bail, hostage, plight, authentication and receipt, just to name a few. And what about the security you enjoy in knowing that you make

regular backups? You do make regular backups, *non*? Then, we have job security, the kind of security you experience knowing you make regular backups. A little joke, *non*?

In all seriousness, losing a file can still happen, and what if your last backup was last night or even two hours ago? Despite good habits and a healthy attraction to job security, you still have lost your file. What can you do? Common wisdom, as they say, informs us that if we delete a file from our Linux systems, that is all, *mes amis*. It is gone forever. However, that is only partially true. It is possible to bring back a file. Observe.

On my test system, I have a small drive mounted on `/mnt/tinydrive`. If I use the command

```
df /mnt/tinydrive
```

I get the following information:

```
/dev/hdd6 42913 9063 31634 22% /mnt/tinydrive
```

I told you that it was small, *non*? On this small drive, I created some directories and files. One file is called `cabernet`, and it contains my family's Cabernet secrets. Actually, it does not contain any such thing, but it sounds so mysterious. Speaking of Cabernets, François has returned with the Burgundy.

Now, if I list this file, it shows up like this:

```
$ ls -l
-rw-r--r-- 1 marcel wines 77 Jun 21 04:38 cabernet
```

At 77 bytes, you know the secret is elegant. Now, if I remove this file with **rm cabernet**, it is gone for good, and it cannot be read. And yet, with a little perseverance and luck, you can bring the file back as I will show you. It is important to unmount the drive in question immediately (or shut down if the file is on your root drive):

```
umount /mnt/tinydrive
```

The more time that elapses (and consequently, the more data that gets written to the disk) from the time you delete a file, the slimmer your chances of recovering it completely.

In order to attempt a rescue of the data in question, I use a program called `debugfs`. Have no fear, *mes amis*. You will find this command on your system. It is part of the `e2fsprogs` package:

```
# debugfs /dev/hdd6
debugfs 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
debugfs:
```

If I type **lsdel** at the debugfs prompt, I will get a listing that looks something like this:

```
7665    0 100644  5248   6/   6 Thu Apr 19 18:05:30 2001
   32  500 100600 12288  12/  12 Thu Jun 21 04:38:39 2001
  158  500 100644    78   1/   1 Thu Jun 21 04:38:39 2001
  157  500 100644    77   1/   1 Thu Jun 21 04:40:25 2001
```

The first column is the file's inode. The second column is the UID of the owner. Since my UID was 500, and my accidental deletion occurred on Thursday the 21st, it is starting to look like my file might be one of those listed. Furthermore, I recall that my file was 77 bytes long. There are no filenames listed so as you can imagine, the more details you know about the original file, the better. Still at the debugfs: prompt, I enter the following:

```
debugfs: dump <157> /home/marcel/cabs
```

The 157 is the inode number, and /home/marcel/cabs is a file on another (still) mounted filesystem. By using the dump command to debugfs, I can redirect the contents of inode 157 to a new file called cabs. Now, let us look at the results of this operation:

```
$ cat /home/marcel/cabs
Why try making fine Cabernets when you can buy them
from Henri's Fine Wines?
```

Mais non! The secret is out!

Another tool you might find useful to restore those lost files is a little program called recover. Written by Tom Pycke, recover makes the whole process of trying to work your way through what can be an incredibly long list of deleted inodes quite a bit easier, one might say virtually painless. To get a copy of recover, head on over to recover.sourceforge.net/linux/recover. Download the latest source, then extract and build it. It is all very easy:

```
tar -xzvf recover-1.3.tar.gz
cd recover-1.3
make
make install
```

To use the program, you can simply type **recover**, and it will scan for the various filesystems available on your system. You can also have it start with the appropriate filesystem by specifying it on the command line. What follows is a question-and-answer session designed to narrow down your deleted file without having to go through a huge listing. The recover program massages the data for you. Here is an example:

```
# recover /dev/hdd6
Recover v1.3 by Tom Pycke <Tom.Pycke@advalvas.be>
```

```
Getting inodes (this can take some time)...
debugfs 1.14, 9-Jan-1999 for EXT2 FS 0.5b, 95/08/09
In what year did you delete the file? (eg. 1999): 2001
```

The program then asks for the month followed by a date and time range as well as a range of possible file sizes. When everything had been answered, I received the following output:

```
=> 158 78 JUN THU 21 4:38:39 2001
=> 157 77 JUN THU 21 4:40:25 2001
2 inodes found. Where shall i dump them?
(directory): /tmp
```

The files will get named dump157 and dump158. All that is left is to view their contents with **cat**. On the off chance that these may be binary files, it might make sense to use the file command first.

Security has another side as we well know (and as Monsieur Roget figured out as well). We may be secure in the knowledge that we can recover a deleted file from our ext2fs filesystem, but what of the other side? What if the information in the file was of a top-secret nature, and I did not want to risk having anyone ever see this file again? What if I would rather have this file disappear into some dark region completely inhospitable to proper viticulture? There could be implications of international security should this information get out.

If your system contains a recent version of the fileutils package (which contains programs like cp, mv, etc.), the solution is at hand. You will find a program called shred that will take your file and, ahem, shred it:

```
$ shred secret_instructions
```

If I use the command as above, the file secret_instructions will be secret indeed, but it will not be deleted. It will, however, be scrambled, much like a well-whisked omelette. For the truly security-conscious, it may be better to use this version of the command:

```
$ shred -z -u secret_instructions
```

In addition to scrambling, the program will overwrite the file with null characters (the -z flag), then remove it completely (-u). Your secrets are safe, *mes amis*.

We started today's culinary exploration with a slight hint of backups. Let us say then, that you archive your data to tapes regularly (a very good idea—I must commend you, *mes amis*). Those tapes are then moved safely to off-site storage. You may well have backed up those secret files that you so desperately wish to remove from your system. Let us pretend that I have archived the /etc/ profile file using tar:

through this archive again (*sans* keyfile of course), what we see is garbled, unreadable and *certainment* unprintable.

Once again, *mes amis*, the time, she is late and we must all start thinking about going to our respective homes. You too, François. But before you do, please refill our guests' glasses. All this intrigue has no doubt done little to slake their thirsts. Unlike Monsieur Bond, pour the wine gently, neither stirred nor shaken. Wonderful. *Merci*, François.

Mes amis, it has been a pleasure once again to serve you here at *Chez Marcel*. Do remember to join us again next time. François will have your table waiting.

A votre santé! Bon appétit!

Resources



Marcel Gagné (mggagne@salmar.com) is president of Salmar Consulting, Inc., a systems integration and network consulting firm. He is the author of *Linux System Administration: A User's Guide*, available September 2001 from Addison Wesley. You can discover lots of other things from his web site at www.salmar.com/marcel.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

PGP: the Best Free Crypto You Aren't Using, Part I of II

Michael D. Bauer

Issue #89, September 2001

An introduction to the underappreciated, 100% free utility you didn't know you needed (but do)--GnuPG.

Ten years after Phil Zimmermann released PGP v.1.0 (Pretty Good Privacy), PGP has evolved from an underground tool for paranoiacs to the gold standard, even an internet standard, for e-mail encryption. GnuPG, the GNU Privacy Guard, is a 100% free alternative to commercial PGP and is included in most Linux distributions. And yet, not nearly as many people who need it (and already have it) use it.

Are you among the many GnuPG procrastinators of the world? Hopefully you won't be after this and next months' columns. After you've generated your personal keys, sent your first encrypted e-mail and finally verified the security signature of that cool software package you downloaded, you'll be glad you took the trouble to master the multifunctional marvel that is GnuPG.

This month we'll begin with PGP/GnuPG background, concepts and quick-start. Next month we'll dig deeper into file and e-mail encryption, key management and graphical user interfaces.

Some Notes on PGP's History, and Why This Article Isn't about PGP

Back in 1991, when the US Congress appeared to be on the brink of outlawing all private use of cryptographic software, Phil Zimmermann released PGP v.1.0. This originally free tool allowed ordinary users of consumer-grade computer systems to encrypt their personal data and communications effectively enough to thwart even determined and well-financed eavesdroppers (such as, for example, the US government).

Phil Zimmermann's story is important and compelling and can be read in Simson Garfinkel's book and on Phil's own home page (see Resources). But for

now suffice it to say that despite government investigation, patent complications and the tribulations of corporate assimilation, PGP has continued to improve and grow in serving Zimmermann's vision of protecting people's and organizations' privacy worldwide.

When I say that, however, I really mean PGP in the broadest sense, including OpenPGP and GnuPG. The emergence of the latter have, respectively, resulted in PGP's key and message formats becoming an internet standard in RFC 2440, and in providing users worldwide with a completely free and unencumbered (by patents) implementation of PGP.

Although Mr. Zimmermann is, by any reasonable standard, one of the true pioneers of and contributors to open-source software, Network Associates Inc.'s (NAI) product PGP is problematic for Linux users in specific and open-source adherents in general. First and most obviously, commercial PGP runs only on Windows and Mac OS.

Second, even PGP Freeware is free only to noncommercial users, that is, users in educational and nonprofit settings. Third, NAI has decided to reduce greatly the amount of PGP source code it makes available for peer review and public scrutiny, even for Freeware PGP.

This last development apparently contributed to Phil Zimmermann's resignation from PGP Security and has cast doubt on the advisability of fully trusting NAI's version of PGP. Considering the US government's hostility toward PGP and various governmental efforts to require "back doors" in cryptographic software (including key-escrow), it's all too easy to imagine NAI quietly bowing to governmental pressure and creating such a back door. Without public scrutiny of PGP's full source code, we have no means of validating assertions by NAI that this is not and will never be the case.

GnuPG, on the other hand, is a 100% open-source and 100% free package that does most of what PGP does (it lacks certain features such as virtual private networking and disk-volume encryption that are included in PGP Desktop). In a short time, GnuPG has become the preferred e-mail and data encryption tool for Linux users and is packaged with most current Linux distributions. The GnuPG Project is headed, and its code primarily developed, by Werner Koch.

What GnuPG Is and Why You Need It

GNU Privacy Guard consists of a single-binary executable, gpg. Actually there's an additional binary, gpgv, but since gpgv merely provides a convenient subset of gpg's functionality you can still think of gpg as essentially self-contained. Therefore, it's perfectly okay to use the terms GnuPG and gpg interchangeably —I'm going to do so for the remainder of this article. I'm also going to be loose

with the term PGP—rather than the specific commercial product by Network Associates, I'll henceforth use PGP to refer to the protocols, mechanisms and Web of Trust common to PGP, OpenPGP and GnuPG.

GnuPG performs four basic functions to which all of its other functions are supplemental: encrypting data, decrypting data, cryptographically signing data and cryptographically verifying digital signatures. It is also used to create and manage keys, activities that, although subordinate to the four listed above, are absolutely essential in performing those functions securely.

In real terms, this means that people generally use GnuPG to encrypt files, especially e-mail; decrypt mail or files that have been sent to them; digitally sign documents, source code distributions and other electronic files; validate others' digital signatures to determine whether a given file matches its accompanying signature (i.e., wasn't tampered with at any point) and whether the file was verifiably signed by the person who claims to have signed it; and maintain keyrings (key databases) containing their personal key or keys (their secret keyring) and the public keys of their friends, colleagues, business partners, etc., (their public keyring).

Obviously, then, you need GnuPG if you wish to exchange encrypted messages and files with other GnuPG users (and users of other OpenPGP-compliant software). It's also one of your options if you wish to encrypt data that is stored locally but in a not-altogether trusted place, e.g., the hard drive of a laptop computer you travel with and are resigned to the possibility of being stolen.

But even if none of your friends use GnuPG or PGP, and even if you feel that none of your data is worth protecting with encryption, there's still one very compelling reason to learn at least a little about using GnuPG: software-distribution signing. Thanks to several high-profile break-ins at public FTP sites on the Internet in which software packages were replaced with compromised (Trojaned) versions, it has become common practice for developers of security software to distribute digital signatures of their software distributions.

How Public Key Cryptography Works (Abridged Version)

We already covered the basics of public key cryptography (PK crypto) in "The 101 Uses of OpenSSH, Part II of II" (February 2001 *LJ*). But this is such an important and fundamental topic that it's worth discussing as it applies to GnuPG.

Remember that the premise is simple: in PK crypto, everyone has a pair of keys, one public key and one private key. A public key is used to encrypt data intended for the public key's owner and to verify digital signatures created with

the bearer's private key. A private key is used to decrypt data encrypted with the corresponding public key and to create digital signatures.

Public keys, unsurprisingly, are meant to be distributed hither and yon; anybody should be able to send you encrypted data, and anybody should be able to verify whether a digital signature was created by you. Conversely, a private key must be truly secret and closely protected from unauthorized copying or use; only you should be able to decrypt data intended for your eyes only, and only you should be able to create a digital signature purporting to be from you.

Public keys may be distributed any number of ways. It's their integrity, not their secrecy that matters. But you *must* protect your private key by storing it in a safe place, such as a directory that only you have read-privileges on, and also by locking it with a long and strong passphrase that must be entered prior to using the private key (I'll tell you how to do this shortly).

All PK crypto systems have that much in common. It's in the distribution and authenticating of public keys that SSH and PGP/GnuPG (and other PK applications) differ. This is important stuff; no matter how effectively you protect your private key, you've still got problems if nobody can distinguish between your real public key and fraudulent public keys distributed by your enemies. (See the Sidebar, "The Need for Meaningful Trust", for examples of the kind of problems I mean.)

The Need for Meaningful Trust

In SSH the problem really isn't addressed. If your friend Bob wants to be able to log in to your server using his DSA key pair and sends you his public key via e-mail, it's up to the two of you as to how to verify that the key you receive in your inbox is the same as the one he sent. (Most likely, you'll simply call Bob on the phone and read part of the key back to him.)

In GnuPG and PGP, there are mechanisms for validating keys, but it's up to you to use them. These mechanisms comprise the Web of Trust.

The Web of Trust

The Web of Trust isn't hard to understand. If Bob knows Ted and vouches for him, anyone who knows Bob can trust Ted. Accordingly, if Bob signs Ted's public key with his own (Bob's) private key, then Ted can distribute this signature along with his (Ted's) public key so that anyone with Bob's public key can, by verifying this signature, see that Bob vouches for the validity of Ted's public key.

The more people who vouch for Ted and sign his public key, the more likely that someone who receives Ted's public key for the first time can check it against a public key they know to be good.

This even extends to the keys of Ted's signing pals. If Ted sends me his public key, but I know neither Ted nor Bob, then Bob's signature doesn't impress me one bit. But if *Bob's* key is signed by my good pal Alice (whose public key I possess and trust), then I can trust Bob's key and therefore Ted's. Hence the "Web".

By the way, this is a radically different model than that behind public key infrastructures (PKIs) such as Entrust and VeriSign in which a single trusted entity vouches for all public keys. The Web of Trust, while certainly imperfect, has no single point of failure; trust is distributed. (See the Sidebar for an example of VeriSign's trust model failing.)

The Web of Trust's main weakness is that unless large numbers of users take the trouble to sign each others' keys and to check the validity of other keys' signatures, there's no web. Sadly, the phenomenon of solo (unsigned) keys is all too common. But your keys won't go unsigned, will they? (Of course not! Just by reading this article you're already on the road to helping strengthen the PGP/ GnuPG Web of Trust.)

By the way, since I mentioned PKI I should point out that while there are PGP keyserver that are used as public key repositories, these are strictly a convenience. Unlike in PKI, in which by definition any key obtained from a certificate authority (keyserver) can be trusted, in the Web of Trust this is not the case at all. Think of a PKI certificate authority as the county courthouse and PGP keyserver as a club newsletter; you can find out somebody's birthday from either one, but they serve two very different purposes and thus differ greatly in their trustworthiness.

Obtaining, Compiling and Installing GnuPG

GnuPG is, as I mentioned at the beginning, now a standard package in most Linux distributions. As is true of all security software, it's particularly important that you keep current, so even if your distribution includes gpg you'll want to check <http://www.gnupg.org/> from time to time so you know when new versions are released.

Naturally, this web site is where you can also obtain the latest source code release of GnuPG. Should you need or wish to build GnuPG from source, simply download the tarball to the directory of your choice (/usr/src is good), and execute the commands below:

```
cd /usr/src
tar -xzvf gnupg-1.0.6.tar.gz
cd gnupg-1.0.6
./configure
make
make check
make install
```

Note that your tarball's name may be different; as of this writing GnuPG v1.0.6 was current, but by the time you read this it may not be. The same applies to the directory extracted from the tarball.

Note also that the `make check` command is optional and somewhat time consuming, but I think it's useful: it prints out some interesting information about supported encryption algorithms and runs a battery of tests that let you know immediately whether `gpg` built properly. If any of these tests fail, there's no point in running `make install` until you've fixed the problem. In the unlikely event of a failed test, refer to the files `INSTALL` and `README` for clues as to why.

Should `gpg` Be `SetUID=Root`?

You may be aware that in general, running programs with their SUID (set user ID) bit set is to be avoided. The SUID bit, which can be set for each file using the `chmod` command, causes an executable program to run with its owner's privileges regardless of who runs it.

For example, if a program has an `s` instead of an `x` in the user portion of its file permissions (as displayed by `s -l`), and if that program is owned by `root`, then any time that program is executed it will have the same rights on the system as `root`; it will be able to do all the things `root` can do.

Sometimes programs are installed with this bit needlessly set and with ownership assigned to `root`. This, however, is not one of those cases. You really should run **`gpg SETUID`** (`SETUID=root`, since `root` owns `gpg`) in order to mitigate the risk of a hostile user reading memory containing a `gpg` private key or its passphrase.

After `make install` finishes, I recommend that you set this bit with the following command:

```
chmod u+s /usr/bin/gpg
```

Quick-and-Dirty GnuPG: Verifying a File Signature

After you've installed `gpg` (whether from source as described above or from your Linux installation media), you're ready to create a personal key pair and start building your own little corner of the Web of Trust. But I've already reached the end of this month's column, so instead let's do something that

doesn't require us to have a key pair of our own: verifying a signature created by someone else.

As I mentioned earlier, digitally signing a software package has become a popular means of providing end users with a means of verifying that the software they download is the same software its developer put on-line.

The command to verify a detached signature (a PGP signature can either be attached to the file it was created for, or it may be stored separately in its own file) is `gpgv`. If we invoke this command on a signature but don't have a copy of the signer's public key, `gpgv` will return an error. In Listing 1 we see a session in which this occurs.

Listing 1. Sample Signature-Verification Session

Let's dissect Listing 1. There were three commands invoked:

```
gpgv gpa-0.4.1.tar.gz.sig
gpg --keyserver pgp.mit.edu --recv-keys 621CC013
gpgv --keyring pubring.gpg gpa-0.4.1.tar.gz.sig
```

The first time I ran `gpgv` (which you may recall is a stripped-down version of the `gpg` command used for verifying signatures) I simply supplied the name of the detached signature I wished to verify. Had I had the appropriate public key on `gpgv`'s keyring, `$HOME/.gnupg/trustedkeys.gpg`, this command would have succeeded, but I didn't, and it didn't.

In the second command, therefore, I ran the regular `gpg` command with the `--recv-keys` directive followed by the ID of the key I had been told by `gpgv` had been used to create the signature. I also specified that `gpg` should look for this key on the keyserver `pgp.mit.edu`. The key was there, so this command succeeded.

In the third command, I realized I'd just downloaded the key to my default keyring, `$HOME/.gnupg/pubring.gpg`, so I used the `gpgv`'s `--keyring` parameter when I reran it. And this time it worked!

There's only one thing I left out in the example, of course, and that was verifying that the key I took the trouble to download was actually from its alleged owner, Werner Koch. And I did do this—it took all of 20 seconds to do a search on www.google.com for “621CC013 werner koch” that turned up a number of mailing-list postings and web sites on which Werner had included this key ID in his e-mail signature.

Were someone to succeed in hacking the web server from which I downloaded, replacing the file with a Trojan horse or virus and a signature created with a

bogus key, and then posting the bogus key on pgp.mit.edu, their skulduggery would be easily detectable by a quick web search like the one I did. I doubt very much that even the most intrepid evildoer would succeed in removing or altering all web sites, Usenet archives, etc., linking his or her victim's name to keys other than the bogus one. So you see, the Web of Trust can work, provided one bothers to do a little follow-up now and then.

I'm already out of space for this month, but there are plenty more useful things to do with GnuPG that we'll discuss in-depth next time. I hope you won't wait until then to try them out!

Resources

Mick Bauer (mick@visi.com) is a network security consultant in the Twin Cities area. He's been a Linux devotee since 1995 and an OpenBSD zealot since 1997, and he enjoys getting these cutting-edge OSes to run on obsolete junk.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Security Applications

David A. Bandel

Issue #89, September 2001

One of David's pet peeves is security, or rather the lack of it.

One of my pet peeves is security or, should I say, the lack of it. It's a near-universal phenomena. Out of the box, most Linux distributions are about as wide open and easy to subvert as the standard Windows box. And this shouldn't be (though fortunately, it is changing). Anyone who connects to a public network (the Internet) should be responsible for that system and its use. We hold gun owners accountable for how their guns are used. The same should hold true for computer owners. But distribution makers should also help their customers in this regard, and a poorly secured system is child's play to break into. As evidence, a large number of script kiddies are teenagers. These children commit DDoS (distributed denial of service) attacks using IRC bots. They won't be punished even as minors, but they can put an internet commerce site out of business on a whim. All it would take is a properly configured Linux firewall in most homes to prevent this problem. And with the new Netfilter, it would be easy to put something together. The biggest roadblock seems to be educating users that they need this, lest their system be used for nefarious purposes.

Free Practice Management:

<http://www.freepm.org/>

I must say, up front, I am not a fan of Zope, and I was unable to get FreePM running via Apache with the provided instructions. Personally, I would have used Perl, PostgreSQL and Apache. But people tend to program using tools they know, I guess. That said, FreePM is more than worth a look for a medical practice. It is extremely well done even if Zope overly complicates its setup. It has support for everything a medical practice needs, including a prescription database, accounting, patient records and more, all well-tied together. Requires: Zope, Python.

Bugtrack:

www.agstools.com/products/bt.html

Extremely easy to install and administer, this utility is very lightweight. If you need a simple bug-tracking solution for Linux or just about any platform, you might want to take a look at this one. It is simple and easily customizable, and I always choose simplicity. Who wants to read documents for days to get something running? For that much effort, I could write the program myself. Requires: a database (MySQL, PostgreSQL, others with ODBC), Perl, DBI module for the database, web server, web browser.

nPULSE:

<http://www.horsburgh.com/>

Another easy-to-install and use utility, this one allows you to monitor your network through a web browser. **nPULSE** relies on nmap to scan your network for open ports to tell you if services are still running. This means nPULSE doesn't rely on SNMP, but it also means you'll have to adjust PortSentry or similar programs to account for the "scanning" activity. However, you may want to modify the HTML sources on this slightly. I find the black background with white writing a bit difficult to read (but that may just be my monitor resolution: 1280 × 1024, 16-bit color on a 19" screen). Requires: nmap, Perl, Perl modules Net::SSLeay and Mail::Mailer, OpenSSL (optional), Java (optional).

SendIP:

www.earth.li/projectpurple/progs/sendip.html

This tool is a must-have for all firewall administrators. Until the new iptables includes a check function, you'll need some way to test your firewall. The SendIP utility will allow you to do exactly that. You can send an IP packet (TCP, UDP) or an ICMP packet spoofing the source address to see what happens on your firewall. Works with IPv4 and IPv6, so you can test the iptables IPv6 rules as well. Requires: glibc.

Sentry Firewall CD:

<http://www.sentryfirewall.com/>

Now this is some CD. You boot from this CD, and with a few preconfigured files on a floppy (on which you've flipped the write-protect switch to read-only) you have a running firewall. Everything is either burned onto the CD or is in memory. If by some quirk, someone actually does manage to break in to this firewall, all you need is a reboot. All files deposited by the attacker are gone

when the RAM re-initializes. This particular CD is based on Slackware. So if you have an old system with two NICs that will boot from the CD-ROM, you've got a firewall. Requires: system capable of booting from a CD-ROM, a way to burn a CD-ROM ISO image.

File System Saint:

<http://www.insecure.dk/>

Don't get me wrong, I like Tripwire. It was the first, but it's certainly not the easiest thing to set up. What I dislike about difficult things is not just that I'm lazy (I am), but difficult often means someone, somewhere will misconfigure or otherwise nullify all the good the program should do. This program is fast and easy. Not necessarily better, but different. Requires: Perl and Perl module Digest::MD5.

datedif:

download only: buschencrew.hypermart.net/software/datedif/datedif-0.9.1-4.tar.gz

This small C program will calculate the number of days between any two dates. Granted, this has somewhat limited use, but the author does present an example. It is most useful in scripts. How many days until Christmas? Requires: glibc.

motion:

<http://motion.technolust.cx/>

I like this little application. It was designed so that you could connect a (surveillance) camera to your system, run **motion**, and it would take pictures each time the image changed. You can make a movie, even send an e-mail. Yep, great for security. But I (ab)use it differently. I connect my video camera after taking a home movie, let motion grab frames every second. Then I load all the images into xv and delete what I don't want and rename what I do want. Much faster and easier than doing it manually—a lot of options. This is a keeper. Requires: libjpeg, libmysqlclient, libz, libcrypt, libnsl, libm, glibc.

Until next month.



David A. Bandel (dbandel@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Embedded Linux at JavaOne

Rick Lehrbaum

Issue #89, September 2001

In a market flooding with similar featured PDAs, Sharp takes a bold leap, unveiling Linux/Java PDA at JavaOne.

At JavaOne in San Francisco, Sharp unveiled a new PDA for the non-Japan market that features an embedded Linux operating system and a Java application environment. Like many (perhaps most) recently announced PDAs and handheld computers, the device uses Intel's StrongARM system-on-chip processor as its core hardware platform. But unlike the makers of other mainstream PDAs, Sharp has clearly decided to gamble that a combo of Linux plus Java—rather than either Palm's highly popular Palm OS or Microsoft's struggling WinCE—will be key differentiators helping it to carve out a niche in the highly competitive PDA market.



Sharp's Linux/Java PDA with QWERTY Keyboard Exposed



Sharp's Linux/Java PDA with QWERTY Keyboard Concealed

Sharp Corporation, which claims to be the largest maker of handheld devices in Japan, has announced a goal of selling one million units of this new device worldwide by March 2002. In the words of Hiroshi Uno, general manager of Sharp's Mobile Systems Division: "we intend to aggressively combine Java with Sharp's user-interface technology and core devices, particularly LCDs, to develop powerful handheld products in wireless communication, data integration, multimedia and PIM-based applications."

The devices are expected to be available late this year, hopefully in time for year-end holiday shoppers to make those geek-gottahave-gift purchases.

As of this writing, the precise name and model number of the new Sharp PDA was not available. However, sources within Sharp are referring to the new PDA as a member of the company's Zaurus PDA family. Sharp's Zaurus PDAs, while highly popular in Japan, have failed to catch on elsewhere, so this new Linux/Java PDA represents Sharp's attempt to succeed by carving out a Linux/Java PDA niche in the US, Europe and other worldwide markets.

The new Linux/Java PDA, a prototype of which was shown at JavaOne, sports a bright, full-color "quarter-VGA" LCD touch/display, along with a unique built-in QWERTY keyboard. The keyboard, which has small keys, is not something you'd want to do a lot of typing on. However, it certainly makes text entry straightforward.

Here's a summary of the unit's hardware specs:

- 206MHz StrongARM SA-1110 system-on-chip CPU
- 32MB DRAM, 16MB Flash (in the prototype, at least)
- 240 × 320 pixel (¼ VGA) full-color TFT LCD with touchscreen
- QWERTY keyboard (behind sliding lower section)
- CompactFlash and SecureDigital card slots

- IrDA, serial, USB and audio I/O ports

One noteworthy feature of this new PDA is that the keyboard is hidden “inside” the lower portion of the PDA; the bottom section slides down to expose the keyboard. Thanks to this clever approach, the device has a typical PDA size when the keyboard is not in use. Also, the keys usually are not exposed to inadvertent presses.

On the software side, Lineo's Embedix was selected by Sharp as the new PDA's internal Linux OS, along with Tao's Intent JVM and ACCESS' Java-based NetFront browser. Additionally, Amiga has been selected to develop Java-based applications for the device, including things like games, animation and multimedia applications.

A key aspect of Sharp's strategy in adopting Linux/Java as the software environment for this new PDA is to create a vibrant developer community around the device. To encourage this, Sharp has launched a developer web site. Those who decide to register as developers (a quick and simple process) are immediately granted access to more detailed information about Sharp's new device and its associated developer program, including a FAQ about the device and the developer program, detailed specs of the device, discussion forums, information about development tools and documentation and application software and libraries (when available).

Speaking of Linux PDAs

Whatever happened to the G.Mate's YOPY and Agenda's VR3 Linux PDAs? Both companies have ongoing beta-level products available for developers to play with, and both are scrambling to get their products ready for production.

Agenda is currently in an “any moment now” wait-loop, so their VR3 may well be shipping to customers by the time you read this. G.Mate hopes to have a fully released product shipping by November of this year and is obviously racing against Sharp to be the first to ship a high-end, Linux-based PDA. Also, MIZI Research is rumored to have formed a partnership with a major Korean electronics manufacturer, in an effort to bring out a Linux-based PDA/cell phone combo by the year's end.

At this rate, we could be facing an oversupply of Linux PDAs as we enter 2002!

Resources



Rick Lehrbaum (rick@linuxdevices.com) created the LinuxDevices.com “embedded Linux portal”, which is now part of the ZDNet Linux Resource Center. Rick has worked in the field of embedded systems since 1979.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Lessons in Mid-Crash

Doc Searls

Issue #89, September 2001

Doc takes a moment to stop and observe the technology-sector crunch from inside the trenches.

A few years ago I did a lot of work in France. Like most countries outside North America, drivers in France are relatively aggressive. In American terms, they tend to tailgate—as if their cars were sustained by the exhaust fumes of the cars they follow. Multiple-car crashes are not uncommon.

One day my friend Lief, a Swedish dude with a very dry sense of humor, came into a meeting and said, “Did you hear how many cars were involved in the big accident yesterday?”

“No”, we answered.

“Guess”, he said.

“Ten?” one of us replied.

Lief shook his head and made a thumbs-up gesture.

“Thirty?”

His thumb poked upward.

“Sixty?”

Still upward.

“A hundred?”

“Two hundred?”

The thumb stayed up.

“We give up.”

“Two hundred thirty-eight”, he replied.

“Was anybody killed?”

“Nope.”

This story comes to mind when I think about the dot-com crash. I'd guess right now (I'm writing this in late June 2001), we're about 180 cars into this thing. It's now obvious that the crashing will continue, right up to the point when every public- and venture-funded technology company that was never a real business ceases attempting to become one. In most cases, that will happen when they finish burning their investors' money—after crashing their investors' cars while smoking one another's exhaust.

When the “internet economy” was still a high-speed traffic jam somewhere back in 1999, I was at a party in San Francisco. Most of the folks there were young, hip “entrepreneurs”. Lots of all-black outfits, spiky haircuts, goatees and face jewelry. I fell into conversation with one of these guys—a smart, eager young chap I'd met at other gatherings. He was on his second or third startup and eagerly evangelizing his new company's “mission” with a stream of buzzwords.

“What does your company do, exactly?” I asked.

“We're an arms merchant to the portals industry”, he replied.

When I pressed him for more details (How are portals an industry? What kind of arms are you selling?), I got more buzzwords back. Finally, I asked a rude question. “How are sales?”

“They're great. We just closed our second round of financing.”

Thus I was delivered an epiphany: every company has two markets—one for its goods and services, and one for itself—and the latter had overcome the former. We actually thought selling companies to investors was a real business model. We lost sight of the plain fact that any company that needs to articulate a business model doesn't really have one.

This idea—making money by selling a potential hit company concept to investors—had a familiar sound to it. After all, VCs seemed to be looking for big IPO scores the way entertainment companies look for blockbusters. VC-funded companies had more in common with movies and Broadway plays than with

real enterprises that worked toward profits and persistence from day one. In the parlance of the entertainment biz, these companies were really just projects.

In fact, the dot-com business model was scripted a quarter century earlier by Mel Brooks as the premise of his first hit movie, *The Producers*, which is now revived as a hit Broadway musical. Coincidence? I don't think so. Check out this dialogue between the two main characters, down-and-out Broadway producer Max Bialystock and his accountant Neil Bloom:

"How can a producer make more money with a flop than he could with a hit?"

"It's simply a matter of creative accounting...you simply raise more money than you need. If you were really a bald criminal, you could have raised a million."

"But the play cost nearly \$60,000 to produce. You know how long it had run? One night!"

"You see? If you...raised a million dollars you could have put on a \$60,000 flop and kept the rest."

"But what if the play was a hit?"

"Then you'd go to jail. Once the play was a hit you'd have to pay off all the backers, and with so many backers there would never be enough profits to go around. Get it?"

"So in order for this scheme to work, we have to find a sure-fire flop!"

Of course, nobody goes into business intending to flop. But plenty of people went into dot-com flops intending both to get rich quick and live comfortably off the investment proceeds in the meantime. And those proceeds were enormous. At its peak in early 2000, VCs were spending two dozen billion dollars on a per-annum basis in Silicon Valley alone. Over half the world's countries have smaller gross domestic products. Lots of that money went to providing upper-class lifestyles to hot young e-xecutives. Just ask the BMW dealers in Silicon Valley.

But why was the rush so big? Because dumb investors were willing to take all the risk. That's what makes the Bialystock scheme work. In a normal business startup, the risk is shared by investor and entrepreneur.

Think about it. If you want to start a restaurant or a small manufacturing company, you go to the bank and get a loan collateralized by very real assets

belonging to you or somebody willing to take the risk for you (like, say, your friendly Godfather). But if you want to start CoApathetic.com, you get your VCs and other investors to take all the risk. Play it right and all you expose are your time and reputation. Yes, your holdings in the company are exposed, too; but the company isn't really a company in the traditional sense. Again, it's a project: a false-front store on the main street of a gold-rush town. Your main job is to make IPO money for your backers. So you attract attention by publicly burning the millions of dollars your backers put up. You buy other companies, rent out cool office space in SOMA, hire a chief marketing officer to put big bucks into "branding"—or whatever it takes to tailgate with everybody else in that high-speed, dot-com traffic jam. (To mix a few metaphors I can't resist.)

Last week I met an interesting dude at our own garage sale. In his own words he had just finished "failing" his company. The sad thing, he said, was that his company actually had good customers—lots of them, and big ones, too—and a good idea (he told me what it was, and I agreed). His problem was that the software didn't work. And the reason it didn't work was that he wasn't a programmer. He had jobbed out the labor. "I learned you can't outsource your core competency", he said. In the absence of that competency, he shouldn't have been in the business. "You can't get somebody else to breathe for you." That was his first lesson.

His second was discovering who to blame for the dot-com crash. "I blame marketing", he said. Why? Because it's so disconnected from reality—from what customers are actually doing in the real world. Always has been, always will be. Marketing is so delusional, and so good at producing delusions, that it tempted thousands of investors and investees to speed their way into a high-speed traffic jam of companies rushing to make money by selling advertising, of all things (like TV! like billboards!) on the Web—in spite of the fact that there is negative demand for the stuff, as demonstrated by the MUTE button on every remote control sold with every consumer electronic device that carries advertising. Worse, it fooled lots of people into thinking they could sell what the Net made free—especially the efficiencies that the Net itself began to produce before eager investors projected fantasies all over it.

But his third lesson was the one that really mattered. "The Net still changed everything", he said. "In spite of all these failures and all this insanity, it's just fine." Why? Because it's real, and its benefits are real. "The Net is the biggest and best thing that ever happened to business", he said. "Even though you can't really sell it."

Likewise, I submit that Linux is the biggest and best thing that ever happened to software, even though you can't really sell it. Like the Net (which is thick with Linux and other free and open forms of UNIX), it changes everything by giving

us something with vast and far-reaching practical benefits. If your company does anything with software, there's a good chance you can put Linux to good work.

Maybe that's why the big companies that have fallen in love with Linux all went through their IPOs a long time ago.

Doc Searls is senior editor of *Linux Journal* and coauthor of *The Cluetrain Manifesto*. His next book will be *Real Markets: What They Are and How They Work*.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

A Question of Licenses

Larry Rosen

Issue #89, September 2001

Which open-source license should I use for my software?

Which open-source license should I use for my software?

Anonymous

Okay, I'll admit it, I wrote this question myself because I've been asked it so many times I wanted to see it answered in print. As an attorney for open-source companies and projects, I am often requested to select a license (or to bless my client's selection) from among the OSI-approved, open-source licenses. (All the licenses described here are listed on the OSI web site at www.opensource.org).

The question puts the cart before the horse. What drives the license selection process is the client's business strategy, and not the other way around.

Do you intend to make money from licensing the software or from providing ancillary services like installation and training? There's nothing illegal about using a proprietary software license if that's what your business model dictates. Of course, as an advocate of open source, I'd try to convince you that there are many advantages to nonproprietary business models—but the client is the one to make the final decision.

What degree of freedom are you willing to grant to your licensees to modify your software? There are open-source licenses (e.g., BSD-type) that impose virtually no restrictions on licensees; they can modify the licensed software and create proprietary versions without restriction. There are other open-source licenses (e.g., GPL-type, more typically known as “free software” licenses) that require the licensee's modifications to be licensed back under that same license; this “inheritance” characteristic is an advantage if you want your licensees to have to reciprocate if they benefit from your contribution to the

community. There are still other open-source licenses (e.g., MPL-type) that impose an intermediate level of freedom; modifications to individual files containing licensed code must be licensed back, but new files that merely work with the licensed code need not be.

Are you willing to grant warranties that the software will be “merchantable” or “fit for a particular purpose”? If your software is royalty-free, you probably can't afford a warranty. On the other hand, you may want to charge for your open-source software and use the profits to provide a warranty and other forms of service.

Is your software so well known that the main asset you need to protect is your trademark rather than your code? An excellent example of this is Apache. Their license allows you to do almost anything you want with the Apache code, but you'll have to change the name. If you have a trademark to protect, make sure your license contains appropriate terms relating to that.

Have you considered the possibility of dual licensing? The owner of a copyright in a software program always has the option to use multiple licenses. For example, you may want to license your software under the GPL and simultaneously provide a proprietary version for those of your customers who are afraid of the GPL's inheritance features; that unreasonable fear can be treated as a revenue opportunity.

Have you considered using different licenses for different parts of your software? Client software might be distributed under an MPL-like license, but server software might be distributed under a proprietary license. That way, you could make money from the bigger customers that will pay to license your server software and simultaneously build a large customer base with free client software.

Are you trying to protect the code itself or the standards that are implemented using that software? A license like SISSL allows anyone to develop modifications of licensed software as long as the licensee complies with all requirements set out by a standards body; a licensee who elects not to comply with the specification must publish a royalty-free reference implementation of the modifications so that the standard cannot be abducted by another company.

If there are patents that relate to your software, you will have to consider licensing your patents along with your code. You may also want to retaliate against any licensee who takes your free software and then turns around and sues you for patent infringement. Various licenses on the OSI-approved license list take different approaches to this problem. Some include a strong retaliation

clause, others a weaker version that may be less threatening to customers with a large patent portfolio.

This is not an exhaustive list of considerations. You and your attorney should understand your business situation thoroughly before you decide on a license. Even after you answer these questions, you will still need to decide whether to invest in the attorney resources to create your own license or to have your attorney modify an existing license to meet your needs. If you choose to create your own license, your attorney will be able to tailor your license to your unique business requirements. On the other hand, modifications to an existing license may be sufficient. Consult an attorney familiar with your business to advise you.

Remember that your business objectives guide the choice of license. Anyone who ignores your business needs and whose first words to you are “use this license” is the wrong horse to push your cart.

Legal advice must be provided in the course of an attorney-client relationship specifically with reference to all the facts of a particular situation and to the law in your jurisdiction. Even though an attorney wrote this article, the information in this article must not be relied upon as a substitute for obtaining specific legal advice from a licensed attorney.



Lawrence Rosen is an attorney in private practice in Redwood City, California (<http://www.rosenlaw.com/>). He is also executive director and general counsel for Open Source Initiative, which manages and promotes the Open Source Definition (<http://www.opensource.org/>).

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

SuSE Linux 7.2 Professional

Don Marti

Issue #89, September 2001

If you're going to install your first Linux server or take on the task of deploying Linux on the desktop, you could do a lot worse than SuSE 7.2 Professional.



If you're going to install your first Linux server or take on the task of deploying Linux on the desktop, you could do a lot worse than SuSE 7.2 Professional. Key features include a journaling filesystem, a crypto filesystem, an easy-to-use desktop environment and network updates.

The first thing you'll notice is that the new SuSE is a huge collection of software and documentation—more than 1,000 pages of manuals, boot and modules floppies, a folder of seven CDs and one DVD and the obligatory square sticker for the front of the case.

Because SuSE is the first distribution to be available on DVD, I decided to celebrate with a new DVD-ROM drive for one of my test systems. Not having to change CDs for a full install, as with previous SuSE installs, is worth it. As you might expect if you've installed Linux in the past few years, pop in the install disk and up comes a friendly GUI install program. SuSE's is called YaST2, and you'll be seeing it later on when you do system configuration changes. The little comic book "Quick Install Manual" helps guide you through the install process and focuses on resizing an existing Microsoft Windows partition, if you have

one (if dual booting, don't forget to back up your existing files when you're converting from Windows, as no resizer is perfect in all situations).

The other test machine was an IBM ThinkPad with no CD-ROM. I put in a PCMCIA SCSI card, connected an external SCSI CD-ROM drive and booted from one of two provided floppies. The installer prompted for the floppy to load the driver for the SCSI card, and the install was underway quickly. I only did a "default" install on the ThinkPad, and it didn't require switching CDs.

A feature you'll probably want, but that is not covered in the "Quick Install Manual", is ReiserFS. Select "custom partitioning - for experts" to get to the secret hidden chain saw, I mean, journaling filesystem. That's also where you'll find the option to make an encrypted partition, and I made /home encrypted on the ThinkPad.

A few screens later, the installer formats the disk(s) and installs packages in one unattended run—unattended if you don't have to change CDs, that is. You might want to invest in a DVD-ROM drive or, if you have a lot of installs to do, reserve some disk space on a server and install over the Net.

The quality of the documentation, which I only started reading when the first install was well on its way, is outstanding. The network manual thoughtfully covers a lot of first-time Linux projects, such as nameserver and Samba setup, and gives a quick introduction to OpenSSH. There's even a chapter on workplace ergonomics.

After installing packages and rebooting (SuSE prompts for the crypto filesystem password on boot if you have one), up comes a good XFree86 setup utility called SaX. There's a large list of monitors, so you probably won't need to find your monitor manual to type in what refresh rates it supports (save the monitor manual anyway). SaX also has a friendly tool to reposition the image on the monitor screen; I didn't need it for 1280 × 1024, but a couple clicks was all it took to get the 1024 × 768 image placed correctly on the desktop machine. For the laptop install, I picked "ThinkPad LCD", and SaX got it right.

Then it's time to configure the printer, sound and network, and you're done. Don't forget to do an on-line update to get security and other updates, and visit SuSE's mailing lists page and subscribe to suse-security-announce, just in case.

There's also a text-mode install program, YaST1, which is scriptable. You can put a partition scheme, a list of packages to be installed, and almost any system configuration parameters on a floppy, and boot from the floppy to install an entire machine from the network. Unfortunately, this potentially useful feature

is under-hyped: the manual doesn't cover it, and the information on SuSE's support site and elsewhere on the Web is sparse.

The KDE2 desktop is the default, and it, along with some other large projects, gets its own directories in /opt. If you're manually partitioning the system, you'll need to create adequate space for /opt as well as /usr, and this is why /opt is fundamentally evil. Applications should each get a subdirectory of /usr/lib or /usr/local/lib, so that you don't need to play Nostradamus when you're deciding how big to make /usr and how big to make /opt. On a desktop system, you'll be able to get away with just root and swap partitions, so /opt isn't so bad there, but anywhere you want multiple partitions for more space or safety, /opt makes you waste space unnecessarily.

SuSE does an excellent job with packaging and integrating KDE. Konqueror, KDE's web browser, is a usable daily driver for all but the worst-designed sites (it even supports a Shockwave Flash plugin), and the rest of the desktop environment is well laid out with sensible defaults. Some of the colors for "ls" are too light to read easily on the default white terminal windows (All right, who at SuSE used to work for *Wired?*), but if you do a lot of command-line work, you'll be customizing that kind of stuff anyway.

Changing system configuration from the desktop is pointy-clicky, with a dialog box to enter the root password for actions that need to be done as root. It looks pretty convenient for a new desktop user and would be easy for a support person to talk someone through.

The number of network ports open on a default install has come down quite a bit, which is a refreshing change from the "everything on by default" policy we've seen in other distributions. KDE and X each have a high port open during a user session, though.

There are "Personal Firewall" (simple block-everything) and "SuSEfirewall" (allow some traffic through) scripts that you can turn on to block incoming connections. A stealth FIN scan with nmap will still see the blocked ports. This is a nice touch for a security-conscious distribution.

There are three more minor issues with an otherwise solid product. First, the last character in the shell prompt isn't set to change to # by default when you **su** to root, which a lot of people expect and a lot of documentation assumes. That's a little confusing. The install should also prompt for an address to which mail for root should go, since you don't want important messages piling up in /var/spool/mail/root. And the "Online Update" feature in YaST2 is nifty but doesn't give you a place to enter an HTTP proxy, which might be the fastest or only way of getting new packages from the Net at some sites.

[The Good/The Bad](#)

[Resources](#)



Don Marti is the technical editor for *Linux Journal*. He can be reached at info@linuxjournal.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

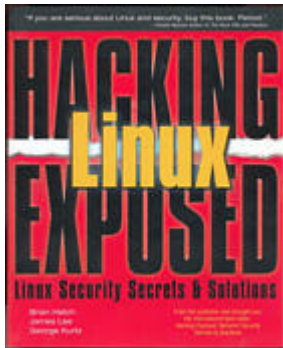
[Advanced search](#)

Hacking Linux Exposed

Thomas Osterlie

Issue #89, September 2001

Hacking Linux Exposed deals with security-related threats to Linux systems.



Past months' Linux worms and the shell server compromise of sourceforge.net, leading to the crack of several open-source sites, have reminded us that not even Linux systems are invulnerable to malevolent attackers. Three leading computer security experts published *Hacking Exposed* in 1999. The book is a thorough dissertation on cracking and was followed up by a second edition in 2000. This year George Kurtz, coauthor of *Hacking Exposed*, teamed up with two leading Linux security experts to bring us *Hacking Linux Exposed*.

Hacking Linux Exposed deals with security-related threats to Linux systems. It's a book for practitioners by practitioners, with an emphasis on practice rather than theory. The book provides an overview of various security-related issues. To this end, it has been organized into four parts. Each part deals with a distinct aspect of systems' security and is in turn broken into several chapters.

Contents

Part I is the system administrator's inside view of systems' security. It deals with how he or she can prevent the cracker from intruding, starting with an examination of the basic security features built into Linux. This is a look at users, groups, file permissions, etc., from a security point of view. While this

should be old news to a systems administrator, this angle into the matter may shed some new light on the topic. The authors then progress to proactive measures and recovering from break-ins. Tools to both search for system vulnerabilities in order to harden a Linux installation and to reveal system compromises are dealt with. Part I is rounded off with a chapter on how a cracker would go about mapping and enumerating your systems in preparation for an attack.

We can make our systems as secure as we want, but there will always be methods of gaining what is considered legal access from the system's point of view. Part II deals with how a cracker could gain such access. Access may be gained in several ways, and the important lesson here is that you can never be too paranoid. Crackers will do anything to gain access, whether it is physical access to your facilities or access through the network. Almost an entire chapter is devoted to social engineering. Worms also receive due attention.

Once malevolent users gain access to a system, their next step will be to elevate their privileges. Local user attacks is the topic of Part III. An entire chapter is dedicated to Linux password systems. For those ever wondering about shadow passwords and PAM, look no further. I particularly like that the authors target attacks against poor programming in this part of the book. Part III ends with an entire chapter on how the cracker can go about maintaining access to an already compromised system. This chapter is particularly useful as it can be read as an introduction to the clues a cracker would leave behind on a compromised system.

While the compromise of a workstation may be bad enough, it is far worse when a server is compromised. Servers play a far more important role in an organization, and server downtime affects more than a single individual. Part IV is devoted to the three major services that Linux supports in both large farms and the kid's bedroom—mail, FTP and Web. General security-related issues are explored along with application-specific issues, including some of the most popular server software like sendmail, postfix, WU-FTP and more. Part IV concludes with a look at access control at the network layer. Both local-access control through the inet daemon and TCP wrappers, as well as external-access control with firewalls are discussed.

The fifth and final part consists of four appendices. The first two appendices, "Keeping Your Programs Current" and "Turning Off Unneeded Services", contain distribution-specific material. Appendix C deals with on-line resources, while the final appendix provides case studies. The case studies are in-depth descriptions of how three crackers have broken into computers.

Presentation

The book is both well structured and well written. It is scattered with gems of computer-security wisdom. I especially like the use of caution and note callouts to emphasize important issues. Each chapter consists of a number of security-related threats to Linux systems, ways to exploit a threat and existing countermeasures. The use of sample scenarios helps clarify the threat and often sheds additional light on the text.

As an aid to understanding the risks involved, all exploits are accompanied by a risk rating. The risk rating is based on the exploit's popularity, how hard it is to perform and the impact it has on the target system. While such figures will always be somewhat arbitrary—it's incredibly hard to come up with any good and exhaustive metrics to measure such factors—the risk rating provides an indication of the overall risk involved with a security-related threat.

A book on computer security would never be complete without descriptions of the tools involved. Both tools to exploit a weakness and tools to fend off and guard oneself against hostile attacks are covered on a per-threat basis. When dealing with the tools, the authors are brief and to the point. This is, after all, a book on computer security as a whole, not a tool tutorial.

I would have expected the authors to explain their use of the term hacking, especially when writing for a Linux audience. We all know how particular some of us are with the hacking vs. cracking issue. In the authors' defense, it has to be said that the original manuscript did contain a section on just this issue, but it was deemed extraneous by the editor and removed.

Conclusion

Hacking Linux Exposed is a good read and a great introduction to computer security on the Linux platform. More than that, it is a great reference work for the more experienced systems administrator. As the authors fairly quickly jump to rather technical material and assume some knowledge of both networking and Linux systems, the book is not ideally suited for the complete Linux neophyte. However, I am of the firm opinion that for a freshman wanting to learn more, the book is perfectly suited as a guide to further reading.

I'm not overly sure I'd recommend the book as a general introduction for those completely new to computer security. For that I'd recommend a book like *Hacking Exposed*, which is a better all-around introduction to computer security. Don't let the two books' similarities in cover and title fool you. There are major differences between them. The problem with *Hacking Exposed* is that it's somewhat lacking in the department of Linux-specific solutions. *Hacking*

Linux Exposed amends this. The two books should be considered complementary.

Hacking Linux Exposed is easy to read. The authors have done a very good job of providing an overview of security-related threats to the Linux platform and how best to avoid falling prey to them. As such, I greatly recommend the book to systems administrators and Linux users who want to learn more about how to secure their systems.

The Good/The Bad



Thomas Østerlie (thomas.osterlie@consultit.no) is a consultant with Norwegian-based consulting company ConsultIT A/S, where he works with server-side systems development for UNIX platforms and with computer security.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Jagged Alliance 2 for Linux

J. Neil Doane

Issue #89, September 2001

This game is something of a classic already, and there's a reason for its popularity despite its potential for complexity; it's really, really, really fun.

Welcome Tribsoft! *Jagged Alliance 2* is the first ported product of this burgeoning Linux gaming company and a great title to select for their debut performance. In *JA2*, a Sir-tech game from 1999, you direct the operations of a force of mercenaries as you attempt to help an exiled leader retake control of his ravaged homeland and release it from the clutches of its despotic ruler. In order to do this, you must manage your own fiscal resources, hire and fire the right mercenaries at the right time, help train guerilla forces, obtain weaponry and supplies, liberate the countryside of an entire nation, and somewhere in there, you have to find time to actually fight the bad guys. As one might imagine, a game with goals so grandiose is going to be a bit long on options and can pack a learning curve steep enough to twist your ankle on, especially if you have an aversion to reading manuals. However, this game is something of a classic already, and there's a reason for its popularity despite its potential for complexity; it's really, really, really fun.

The Story

Jagged Alliance 2 was actually the third installment in the Jagged Alliance series (the second being *Jagged Alliance: Deadly Games*), and actually it isn't the latest. However, Tribsoft is working tirelessly, we're told, to bring us the next *JA2* in the series, *Jagged Alliance 2: Unfinished Business*, at some point in the near future. Basically, the game revolves around the actions of a horde of mercenaries under your control as you attempt to aid the exiled leader of the fictional nation of Arulco. It seems that Queen Deidranna has abolished the democratic government and set herself up as supreme ruler of the tiny country using her military strength to squeeze what wealth and power she can from her own people (I guess NATO's sitting this one out). The deposed leader of Arulco, Enrico Chivaldori, contacts you secretly, and you meet in a quiet bistro at the

beginning of the game where he tells you his sad story and offers you lots of money to help him out with his homebrew revolution. The plan is simple: you go in, take back his cities, train his people to defend themselves and restore him to power. If you agree to help, he can provide a network of contacts and enough money to get you started (with the promise of more later). Well, of course you agree (you're a money-grubbing merc and he has a briefcase full of money for goodness' sake), and the adventure begins.

The internal story is really nonlinear. Though the ultimate goal of retaking the country remains paramount throughout the game, you are out there to help the people of Arulco, and as such, are often tasked with subgoals of various types. Objectives like finding wanted terrorists for bounty, giving messages to covert agents, training militias and clearing supply corridors for rebel forces are the kinds of tasks you'll run across at every turn in *JA2*. The map itself is so complex and detailed that you'll also find smaller goals hidden inside larger ones from time to time. For instance, during one phase as you clear out a sector of countryside of enemy forces, if you're looking inside buildings closely you might notice a small T-shirt company. Inside, you'll discover that the place is exploiting children for labor, and you can free them with the right plan. During the game, you actually can talk and interact with NPCs (nonplayer characters—there are over 150 of them); indeed, they're invaluable sources of information and add much to the development of your game's own unique storyline and history. Perhaps the only downfall of the free-form nature of *JA2*'s storyline is that it opens the possibility for some very long, long games, and while people simply can play small portions of a larger campaign to get their fix between *The Simpsons* and *The X-Files*, there really isn't any such thing as a short *JA2* game.

Game Play

Jagged Alliance 2 is a bit of a puzzle. Is it a role-playing game? No, well, yes. Is it a strategy game? Sometimes. Is it real-time or turn-based? Yes, it's both; it depends—okay, *JA2* is obviously a bit hard to pigeonhole into a single category. With *JA2*, Sir-tech blended the best parts of strategy, role-playing, real-time tactics and turn-based gaming in a clever balance that makes the game really flow nicely once you get into it. The getting into it, on the other hand, can be something of a challenge. The diversity and complexity of *JA2*'s control interfaces can make *SimCity* look tame by comparison; however, I think most players would agree that once they master the basic elements of the game and get a certain feel for the minutiae of the interfaces, the level of control is not only necessary for the combination of role play, strategy and tactics to function properly, but actually provides a rather comforting degree of flexibility and adaptability in the game's mechanics.



Your Eniac Laptop!

The game starts off on your merc's laptop. Sir-tech has created its own faux OS interface, loosely based on Windows it would seem, called sirOS VIII, which can connect (via satellite modem one must assume) from deep in the jungles of Arulco to a Sir-tech mockup of the Internet. Much of the resource management in the game is initiated from here. Indeed, your laptop is nothing short of vital to your success as a mercenary; you send and receive e-mail from it (and even receive spam that says it's not spam), you browse trade web sites, purchase things (like guns and ammo) from e-commerce sites (some of which are occasionally under construction) and balance your budget on it. You store files on it, such as maps of Arulco and reconnaissance information, and can even browse your logs to recall what you've done and when.

The game gets going when, from your laptop, you connect to a couple of web sites and recruit your initial team of mercs. A.I.M., the Association of International Mercenaries, is the largest and most prestigious job exchange available to you, though you do have the option of personalizing and creating your own custom soldier, a mercenary who works for free in your group. Your only limitation on hiring mercs is money, and each merc has his or her own dollar amount.

What's more, each merc is totally unique—they have personalities, they have their own equipment, some get along well with others, some don't, some are true professionals who have been doing this all their lives and some are just crazy backwoods freaks who want to kill someone for money. One of your biggest challenges in the game is to construct a team of mercs that will get along with each other, compliment each others' abilities well, and most importantly, be able to fight their way through the evil Queen Deidranna's forces and do it all within your limited budget. As you progress in the game, you'll have the opportunity to hire or recruit some of the more advanced mercs and the mercs you hire now will learn new skills to become wise veteran soldiers.

It's interesting actually, to get accustomed to your team and see their individual potentials and personalities emerge and grow as you play. Some can be quite crass (parents be warned: some mercs curse like sailors and there are no parental controls available), others can be wimpy and nerdy or shy and introverted, and almost all of their random dialogues will leave you grinning and shaking your head. One of my favorites was a merc code named Steroid speaking in his Hans and Franz-style Schwarzenegger voice to announce that he was wounded with a deadpan "My skin is punctured and leaking."

Your mercs are dropped into a city in Arulco, and basically from here on, you move around the map in real time. If it takes you two minutes to cross a bog, it really takes you two minutes. The large map of Arulco is divided up into large parcels of land, or sectors, that make up the map your mercs are on at any given moment. You can separate your mercs into as many squads as you like and send them to as many different sectors as you like (in fact, as the game progresses, you'll leave mercs all over the place to train the locals to be militia soldiers), but you can only zoom in to look closely at one sector at a time. If you like, you can exit into a control screen where you can speed up time to make those long six-hour journeys from one part of the country to another flash by in the blink of an eye, monitor the locations of all your mercs on one screen and zoom in at will to any particular hot spot that interests you. If your mercs run into an enemy at any time, whether in the close-up real-time view or in the overview, or Map Screen mode, the game immediately switches over into turn-based combat, and you have the options of either retreating your people to an adjoining sector (if you're near a border) or duking it out with all the Queen's men in the sector.



Shoot-out at the Airport Gates

In turn-based combat, your mercs have dozens of movement options available to them, and you must use combat tactics to maneuver and manipulate them appropriately to win the battles. You can use the terrain or foliage to provide cover or even vantage points for ambushes. Your mercs can run across the

map to new locations, crouch down, crawl or even go into a stealth mode that you can use to sneak up on enemies who haven't seen you yet. Should your mercs be wounded in battle, someone in your party with medical skill must administer first aid to them or they will continue to bleed and lose health. Your mercs can take and give damage to various body parts (leg shots tend to make them fall down temporarily, head shots can cause serious injury, etc.) and even fight hand-to-hand when close enough to an enemy.

Outside of combat, your mercenaries are still central to your game play. With your mercs, you secure your only source of income in the game, the gold and silver mines of Arulco. The more of these mines you can secure with your forces, the more monthly income you can derive from them, and thus the more mercs you can hire and sustain. Mercs also manage weaponry and supplies, they carry and find all sorts of guns, ammo and armor around the countryside, as well as medkits, lock-pick kits, crowbars, canteens, flak jackets and a host of other weird sundry items like bubble gum or beer. Mercs can also train militia or practice skills when left alone and actually need to sleep occasionally or will become fatigued and unable to fight well.



Move Your Troops from Sector to Sector on the Map Screen

The graphics in JA2 are, well, they're from 1999. The resolution is limited to 640 × 480, and though I'm not sure you'd actually need to go higher to get any more information, the display seems somewhat pixelated and blocky by today's standards. The flip side of this coin is that you quickly forget about the graphics once you start getting into the game play or realize that 3-D-rendered bushes aren't really required for this type of game. The environment is phenomenally complex and epic in scale; moreover, thousands of objects in each sector can be manipulated, opened, closed, picked up, blown up, etc. Half the fun of playing is exploring the diverse landscape and poking around in buildings for loot. The character animation is really excellent (the close-up combat scenes actually reminded me quite a bit of Maxis' *The Sims*--the detail on the character movement is just that good), and the cutscenes and "movies" are drawn with

the game engine itself. The only real fault I could find in the graphics engine is that the terrain modeling is almost impossible to interpret in complex situations—it's nearly impossible to tell if you are shooting over or at obstacles between you and your targets (who, incidentally, could apparently see through walls or trees from time to time). The sound effects are really awesome, and the musical score, while perhaps somewhat repetitive in a game that takes this long to play, always fits the mood of the action perfectly.

Unfortunately, there simply is no multiplayer mode for *Jagged Alliance 2*, period. However, at the start of each game, you do have several options available to liven up the party. The difficulty levels do make a big difference in this game, and your gaming experience changes dramatically based on which level you choose. You can also set a Tons of Guns option that drastically increases the numbers of types of guns available to you at any given point in the game, and you can even, get this, hit the Sci-Fi option wherein huge alien insects will attack your mercs at night!

Linux Notes

Jagged Alliance 2 requires a fairly minimal hardware setup to run. Tribsoft recommends only a PII 233MHz with 32MB RAM, a 4X CD-ROM and 400MB of free hard disk space. Since nothing about *JA2* requires 3-D acceleration, pretty much any card that will run at least XFree86 3.3.x at 640 × 480 or higher should work just fine. An OSS-compatible sound card is required, of course, and Tribsoft also suggests a version 2.2 Linux kernel (or greater) as well as glibc 2.1 or higher. The game played like a dream on both of the 700MHz and 500MHz systems I tested it on (GeForce/Debian woody and G400/VA-enhanced Red Hat, respectively). The installation footprint can vary from 305MB (the base installation) to over 800MB (if you want to put all the maps and speech files on your hard disk) and can be done with either a shell script or a nice-looking graphical installer.

Summary

This is really a great game. At first, I admit, I was skeptical about this title, owing mainly to its relative age in the gaming market, low-tech graphics and seemingly overly complex controls, but once I got to know *JA2*, I really began to understand just how cool this game is and why Tribsoft chose to port it. It's one of those games that you can play a hundred times and have a hundred different games and still want to try something different the next time. There's an attractiveness to this game that's hard to define; maybe it's the profound individuality of the mercs themselves, the personal touches from all the dialogues of the NPCs, the choose-your-own-adventure feel or the ever-so-smooth transitions between strategy, tactics, real time and turn-based action

modes. Either way, *JA2* is a game that you can play for hours on end and still want more. Highly recommended for anyone not afraid of a little complexity in their games.

[The Good/The Bad](#)



J. Neil Doane (caine@valinux.com) is an engineer with VA Linux Systems, and between prolonged spasms of geekness, hardware scavenging and video gaming, he is a pilot, guitarist and very poor snowboarder.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Letters

Various

Issue #89, September 2001

Readers sound off.

Praising Wayne

I'd just like to pass along my praise for the article "Algorithms in Africa" by Wayne Marshall in the June 2001 issue. This is a definite upgrade over the typical Linux success story. Insightful, committed, poignant, experienced and informed, you should consider Mr. Marshall's perspective as paradigmatic for covering the emerging global presence of Linux. The principles and values here have definite application to domains such as India (where the FSF is opening a branch office), China (where the government has adopted free software, if not political freedom, as its own) and many other areas of the world such as Eastern Europe and South America. Please keep us up to date on global development. And thanks for a superb magazine.

—William G. McGrath

Hurrah for Us

I just wanted to write to let you know that I've consistently found *Linux Journal* to have the highest quality content in the magazines that cover Linux and technology. I am constantly getting refresher courses, learning about new code and projects, and generally getting fantastic info from your publication.

I especially wanted to compliment you on your regular sections: At the Forge, Cooking with Linux and Paranoid Penguin. Much of the information is applicable to other *nix to boot, making your publication one I keep around for a long time (much to the consternation of my wife). Anyway, thanks folks, and keep up the good work.

—Joe Heck

Unchecked Checker

In the July 2001 *LJ* article "Debugging Memory on Linux", I noticed that the open-source memory checker I've been using was not listed in the article. The checker contains a replacement malloc library plus patches for gcc. The gcc patches wrap C++-like constructors around local variables and insert tests before memory references. This allows checked programs to detect memory overwrites of local variables and some global variables in addition to malloced buffers, and the checking catches overwrites as soon as they happen. You may freely mix object modules compiled with and without checking. The checker also includes replacements for mem* and str* routines and can detect invalid calls against checked memory objects, even from modules compiled without bounds checking.

There are links to the checker from the gcc extensions page at gcc.gnu.org/extensions.html.

—William Bader

Proprietary Parity

In your article "Debugging Memory on Linux" in the July 2001 issue of *LJ*, you list Purify from Rational as a proprietary tool. As far as I can tell from their web site, they do not support Linux. Also, a while back I did talk to a Rational salesperson who said they didn't have any plans to support Linux. Do you know something else?

Sorfa replies: It looks like you are right. At the time of writing the article (early this year), there was a hint that Purify would be supported on Linux. I assumed (wrongly) that by the time the article made it to press, it would be available. It is a pity and I apologize for the incorrect info. It looks like the only proprietary alternative is Insure++.

Certification Revisited

I must respectfully disagree with Allan Hall in his letter of the July 2001 issue. Certification per se is certainly no substitute for experience, but it does show that a candidate at least took the initiative to attend some classes, read some books and pass some tests. It also usually requires putting a few hundred dollars up front.

I don't see how one could give a certified candidate anything but an edge over an uncertified one, experience levels in the two being equal.

—Bill Cunningham

MPlayer Magic

Just want to write to let you know that Robin Rowe's article "MPEG-1 Movie Players" (May 2001) was very helpful and also convinced me to renew my subscription to *Linux Journal*. I wanted to play movies on my new notebook and had played with xanim before, but your recommendation of MPlayer was great. It compiles, installs and works like a charm. Thanks again.

—Chuck

Kinder, Gentler CVS

It's articles like "CVS: an Introduction" (July 2001 issue of *LJ*) that keep me subscribed to *Linux Journal*. I've been doing basic RCS for years and knew there had to be a better way. But let's face it, the man page for CVS is a little overwhelming to the uninitiated. But the day after reading the article, I was using CVS at work (the magazine is opened on my desk to page 72 right now), and I'm feeling much better about long-range management issues now. Keep 'em coming! So many thanks to you and Ralph Krause for putting this together.

—John Evans

Praise from the Source

I just went through your article about Nessus in *LJ* (June 2001), and I just wanted to congratulate you, both on the accuracy of what is being said, as well as on the excellent step-by-step instructions you gave to readers.

Just to illustrate how well you explained: I only received a few installation-problem questions from (what seems to be) readers of your article, while the mailing list gained about 100 new users the two weeks after the release of *LJ*. (And congrats for not having taken the screenshots directly off nessus.org, as so many persons do.) A very fine job indeed.

—Renaud Deraison The Nessus Project

Think Again

Choong Ng's review of Mandrake 8.0 in the August 2001 issue gives a good overall picture of the strengths and weaknesses of the release. However, he failed to mention one important issue. It seems that the stock kernel shipped with Mandrake 8.0 will not work with the PS/2 mouse pointer in many IBM ThinkPad laptops. On the ThinkPad mailing list, some people have not even been able to install Mandrake 8.0 because of this.

—Mendel Cooper

Errata

Izzet Agoren's Kernel Korner (May 2001) The line of code on page 28 that reads

```
echo /proc/sys/net/ipv4/ip_forwarding
```

should read

```
cat /proc/sys/net/ipv4/ip_forwarding
```

Mitch Chapman's "Create User Interfaces with Glade" (July 2001) The link for the Glade home page listed in this article is obsolete. The new location is glade.gnome.org.

—Editor

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

UpFront

Various

Issue #89, September 2001

Stop the Press, *LJ* Index and more.

Penguin Knights

For reviews of Linux chess interfaces see <http://www.firstlinux.com/articles/chess/>.

LJ Index—September 2001

1. Percentage of computers that have been kicked or mauled by their users: 25
2. Size in billions of dollars of Microsoft's cash hoard, as of May, 2001: 30
3. Rate in billions of dollars/month at which Microsoft's cash hoard is growing: 1
4. Percentage increase in Microsoft's stock price in 2001 through May 30: 62
5. Number of possible simultaneous conversations possible when Marconi made his first xmission: 1
6. Number possible now, in trillions: 1
7. Years in which this number doubles: 2.5
8. Number of square kilometers of land in the world: 148,940,000
9. Number of simultaneous conversations per square kilometer: 6,714
10. Number of pages Google finds containing the phrase "open source": 1,930,000
11. Number of pages Google finds containing the phrase "free software": 1,150,000
12. Number of pages Google finds containing the phrase "Eric Raymond": 26,100
13. Number of pages Google finds containing the phrase "Eric S. Raymond": 30,100

14. Number of pages Google finds containing the phrase "Richard Stallman": 54,300
15. Number of pages Google finds containing the phrase "Richard M. Stallman": 11,600
16. Number of pages Google finds containing the phrase "Copyleft": 176,000
17. Number of pages Google finds containing the phrase "GNU/Linux": 446,000
18. Number of pages Google finds containing the word "Linux": 26,500,000
19. Linux shipments as a percentage of all server shipments in Q3 2000, according to Gartner: 9
20. Linux as a percentage of the total server market, according to IDC: 27
21. Percentage of respondents who say they are already using Linux, according to AllNetResearch: 39
22. Percent IBM Linux revenue growth: 128
23. Percent revenue increase for Linux shipments from 1999-2000: 28
24. Projected Linux server installed base in 2005: 21,006,000

Sources:

- 1: *Wired News*
- 2-4: *TIME Magazine*
- 5-7: Martin Cooper, CEO, ArrayComm and inventor of the mobile phone
- 8: *CIA World Factbook*
- 9: mathematics
- 10-18: Google, June 11, 2001
- 19: Gartner Group
- 20: International Data Corp.
- 21: Internet.com
- 22: IBM
- 23-24: International Data Corp.

```
## _A_Lug's_Life_ - (c)Dave Edwards
<amoamasam@sympatico.ca> 2001
_Minutes of the General Meeting of FOOLUG (Formerly
the Oxbridge and Orford Linux Users Group)[1],
Aug. 2 2001_
Convened and brought to order at 7:05PM by Joseph
Liebe, meeting coordinator, in room 110 Brandt College.
Present:
  Joseph Liebe, meeting coordinator
  the board of directors--
    George H. Walker, President
    Ravi Singh, Vice-President
    Christina Howe, Treasurer
    Rick Joiner, Secretary
  and 43 members.
1. o _Linux Can Conquer Cancer_ (LC3)
   presentation by Mike Kelly, followed by
```

- Q&A = 40 m.
 - 2. o _Introduction to XFree86 and X11R6, Part XIV_ by Ewen = 3 m.
 - 3. o General open Q&A = 20m
- Motion Raised:
- o by Alf Tupper
 - o that the 3 Tux plush dolls received by FOOLUG from Eazel(r) (with the "Eazel(r) Forever" logo on their chests) with the sample CDRoms be distributed as prizes to the top finishers of FOOLUG's Code Wars 2001.
 - o seconded by John Combe, top finisher in FOOLUG's Code Wars.
 - o put to show of hands
 - o carried
 - o Christina inquired re the whereabouts of the Eazel(r) Tuxes
 - o minutes of March meeting of FOOLUG board of directors consulted
 - o "Eazel(r) Tuxes entrusted to Christina"
 - o # find / -name 'Eazel(r) Tuxes'

Meeting adjourned @ 8:30PM
 20-odd members adjourned to local for FOOLUG Beerswill
 * * * * *

Re: Tux took a hike
 Date: Mon. 11 June 2001 19:48:03 -0400 (EDT)
 From: Mal Tremblenc <mtrem@xxx.xxx>
 To: foolug@lists.foolug.lug
 Reply to: foolug@lists.foolug.lug
 On 07-June-2001 Christina Howe wrote:
 <snip>
 > the Tuxes must have been swiped somewhere between
 > the end of the March meeting and the end of the
 > beerswill, I can't remember.
 Yah, I believe that. ;))
 > But I want them back. They're practically
 > collectors items--like the Spruce Goose or
 > something like that. So whoever took them, please
 Only penguins, and smaller, and not as sprucey.
 > return them. I've arranged with the staff at
 > Brandt College for the reception people to accept a
 > package an hour before the July meeting, no
 > questions asked. That's fair, isn't it?
 > Give someone else a chance at them. Not
 > *everything's* free, you know. Most directly,
 > the thing you do is theft.
 <snip>
 Hmmm. That rings a bell. Anyway, do like the lady
 says. I want mine.
 Mal.

--
 It is always easier to ask forgiveness than it is to
 get permission.
 +++-+-+--+-+--++
 Re: Tux took a hike
 Date: Mon. 11 June 2001 19:51:34 (EDT)
 From: Justin E. Cohen <jec@xxx.xxx>
 To: foolug@lists.foolug.lug
 Reply to: foolug@lists.foolug.lug
 On 07-June-2001 Mal Tremblenc <mtrem@xxx.xxx> wrote:
 8<
 > > Not *everything's* free, you know. Most
 > > directly, the thing you do is theft.
 > <snip>
 > Hmmm. That rings a bell. Anyway, do like the lady
 > says. I want mine.
 Me too.
 Justin.
 +++-+-+--+-+--++

[1]This new name was chosen by majority rule at the meeting of May 3, 2001, after the townships of Oxbridge and Orford were merged into the new Municipality of Orbridge, as a compromise in order to please both sides of a divisive flame-war.

Hacking the NIC

In the February 2001 issue of *LJ*, Bill Ball reviews the NIC (New Internet Computer) and points out the flexibility and cost of the NIC, which offers a “tantalizing opportunity for Linux hardware and software hackers”. Since then at least a couple of you have risen to the challenge. Look for an article by Jay Sissom on booting the NIC from a network in the September/October issue of our sister publication, *Embedded Linux Journal*. Also, for a HOWTO on replacing the NIC's Flash memory with a 2.5" laptop hard drive see www.virtualdave.com/~dpsims/NIC-server/nic_review.html.

DISA Adopts StarOffice

On June 25, 2001 Sun Microsystems announced that the US Defense Information Systems Agency (DISA), which manages command, control, communications, computing and intelligence systems for the Department of Defense, has adopted 25,000 units of StarOffice 5.2 productivity suite. DISA will replace Applix, which it currently uses, with StarOffice 5.2 software on over 10,000 UNIX workstations at 600 client organizations worldwide.

OS Study

In the first quarter of 2001 a study was conducted to discover which operating systems electrical distributors in the southeastern United States used. This study examined the types of operating systems in use, how they were used, the perceived reliability of each operating system and future plans for implementing other operating systems. After developing a survey instrument, 159 surveys were sent and 92 were completed and returned. Responses from the 92 respondents were used for data analysis.

The research revealed that distributors were using 15 different operating systems. The operating systems used were Windows NT, Windows 95/98, Windows 2000, Windows Millennium Edition, UNIX, Linux, OS/400, OS/2, Open VMS, Novell, VSE/ESA, MPE, Solaris, DOS and Advanced 36. Of all 92 respondents, 78 used Windows NT, making it the most widely used operating system. Windows 95/98 followed close with 77 respondents. Only 17 used UNIX and 6 used Linux.

With all operating systems included, the most common use for the operating system was as an application server. Windows NT accounted for 46% of the application servers. The second most common use was a fileserver, with Windows NT accounting for 64% of those. Print servers, e-mail servers and web servers, respectively, were the next most common uses for the operating systems. Windows NT was the most common operating system used for each of these purposes.

Overall, although not as popular, non-Microsoft operating systems were rated significantly more reliable than Microsoft operating systems. The mean reliability score of Microsoft operating systems was 7.62, while the mean reliability score for non-Microsoft operating systems was 9.66. Windows NT was the most reliable of the Microsoft operating systems. Of the non-Microsoft operating systems, OS/400 had a mean reliability score of 10 (n = 6), followed closely by Linux with a mean score of 9.4 (n = 6) and UNIX with a mean of 9.36 (n = 14). For a PDF version of the full study, see <http://www.jcpb.com/ospaper/>.

—John Williams

They Said It

Markets can remain irrational longer than you can remain solvent.

—John Maynard Keynes

The second oldest profession is bookkeeping.

—Craig Burton

Being afraid of monolithic organizations, especially when they have computers, is like being afraid of really big gorillas, especially when they are on fire.

—Bruce Sterling

I'd doubt that it represents a threat to anything but the notion that you can sell bad code by refusing to let anyone see what's in the box.

—Amy Wohl on Linux and open source

The personal communications industry started with the first portable cellular call 28 years ago.

—Martin Cooper

Culture is the tacit agreement to let the means of subsistence disappear behind the purpose of existence. Civilization is the subordination of the latter to the former.

—Karl Kraus

At the close of the first day the cash drawer revealed a total of \$24.67. Of this sum, \$24 was spent for advertising and 67 cents saved for making change next morning.

—John Wanamaker, after opening what was to become the world's first department store, in Philadelphia, 1861.

Is the Internet world better? Well, Internet has its share of acronyms and techno-babble. But because the Internet is an open platform, because it's free from the monopoly legacy, the entrepreneurs who ply the Internet have the freedom to focus on the real problems, to create and search out markets and then to make those markets work. And, of course, they have the freedom to go out of business when the businesses and markets don't work as they hoped and expected. That's how free enterprise works and that's the real secret behind the success of the Internet.

—Martin Cooper

Genius is an infinite capacity for taking pains.

—Jane Hopkins

ULB in Upcoming Issue

Don't miss our annual Ultimate Linux Box article forthcoming in the November issue. This year's *LJ* ULB gets an upgrade from single to multiple processors by including AMD's new Athlon MP processor. See www.amd.com/products/cpg/server/athlon/partners/partners.html for a few places where you can get yours.

Stop the Presses: Recasting .NET

On July 9, 2001, Ximian did the unprecedented: they embraced and extended the Microsoft "innovation" called .NET. Titled the Mono Project, they called it "a community initiative to develop an open source, Linux-based version of the Microsoft .NET development platform". In a number of interviews, Ximian CTO Miguel de Icaza was remarkably flattering to .NET. To *The Register* he said, ".NET solves a number of problems we've been trying to solve in GNOME. Instead of wasting our time trying to create a new standard, we're embracing .NET and extending it for our own purposes." To *Linux Journal* he said, "We saw in .NET a mechanism that would allow programmers to become more efficient. You have garbage collection. You have thread-based libraries. You can use any programming language—and none of the class libraries have dependencies on Windows. There is a lot of device and platform independence."

On Slashdot and Linux Today the responses ranged from "Way to go" to "Dare we ask it? Has de Icaza been seduced by the dark side?"

In fact, the effort has a lot more scope in ambition than in actual size. While .NET is a vast development framework, de Icaza tells us:

We're mostly looking at just what Microsoft calls the Framework SDK. This basically consists of three pieces. The first is a virtual machine—they call it a virtual execution system—which is basically a runtime. So far we have implemented one-third of our C# compiler; we've started work on a runtime engine, a just-in-time compiler for Linux; and we've also started work on class libraries to run our compiler inside of Linux. It's not a complete project yet and won't be for another six months. Then we hope to have by the end of the year something that can run the compiler—a system which is self-hosting, so we can compile the compiler itself on Linux. It's a huge project. That's why it has to be an open-source project. We can't do all the work alone.

Craig Burton, an analyst with Burtonian.com (who joined in during the same *Linux Journal* interview) says, "It won't be easy, especially since Microsoft won't even be done with the .NET framework before the end of the year. Microsoft hasn't even explained .NET to itself. A lot could change before the end of the year."

De Icaza responds, "Microsoft right now has .NET at a point it calls 'API stable'. We're working from (what they) submitted to the European Commission." Instead his main concern, shared with many who remarked on the announcement in public forums, is with licensing. De Icaza thinks Microsoft's public concerns about the GPL are a smokescreen for its perception of Linux as a direct competitor for its service business. Regardless of Microsoft's motives, any attempts to embrace and extend .NET are likely to be a source of legal concern to the company.

Meanwhile the idea in the medium-long run, de Icaza said, is to let developers create .NET applications and run them on any Mono-supported platform, including Linux, UNIX, and of course, Windows. That's the goal he and Ximian will be working toward with the Mono Project.

On Slashdot "polar bear" writes:

It's a long shot. But if IBM, Sun, HP and the rest got behind a true open standard Web services framework, Microsoft wouldn't be able to deny its competitors an equal playing field—which is exactly what it wants to do with .NET—they want to deploy pieces of .NET to other OSes to lure people in to using it, but the choice bits will only work with Windows. An open .NET would allow everyone to have an equal footing. Sure, Microsoft could still play ball, but they'd lose some of their bully power.

In the days that followed the announcement, however, another potential strategy emerged, suggested first by Burton—who employed exactly the same

strategy when he was fighting Microsoft, successfully, at Novell in the 1980s—then picked up by others. That strategy is “redirection”. By contrast, Burton calls Ximian's strategy “cloning”. He writes, “The cloning method simply means that you rewrite all of the code that Microsoft has developed and run it on a different platform.” Redirection, he continues, is a way to “redirect” or “hijack” the Microsoft-generated object data flow to servers that are not controlled by Microsoft. The same strategy could apply to data streams of all kinds, including instant messaging (IM). XML streams of AOL and Microsoft Messenger client code could be redirected into Jabber's IM server, for example.

In the long run, Burton believes, “.NET (all of it) and HailStorm are going to be redirected technologies. If I were Ximian—or any other open-source vendor for that matter—I would be busy building redirection-based code, not clone-based code.”

And thanks to the nature of open source, the bazaar will decide.

—Doc Searls

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

Security Begins with Me

Richard Vernon

Issue #89, September 2001

The answer you don't want to hear.

I recently stopped by the Seattle offices of the security consulting firm @stake (the current employer of world-famous Mudge) to have lunch with Frank Heidt, a friend who is managing security architect. I unexpectedly ended up having to wait some minutes while Frank attended a conference call. When he came out, it was to complain about the weekend of work ahead and to tell me that our lunch would have to be a ten-minute coffee break instead.

The conference call had been from a client company who was having difficulty in selecting among the short list of unsavory options presented by @stake. They are the victims of their own security department gone rogue. At this point, at the mercy of their own employees, their choices are few and expensive. Frank tells me that in his experience, a significant majority of security cracks and threats are internal, which reminded me that a majority of murders and rapes are also committed by perpetrators known to the victim. Rather than barred windows, pepper spray and firewalls, the better investment may be in the time you take to choose whom you let in the physical door. As Bob Toxen writes in *Real World Linux Security*, "The presence of a firewall...should not be an excuse to allow insecure systems behind it."

Given that complete security is unachievable and laxity foolhardy, I asked Frank about his security philosophy. He replied that he doesn't really have one specifically, but that the client's requirements should determine the security strategy to be taken. He views security not as a magic list of firewalls, tools and daily tasks (though he believes Snort to be about the best IDS out there) but more of a set of requirements to be met and limitations to be considered. For those looking for that holy grail of security, this seems like a nonanswer, but it's really the only one that makes sense. Apologies for returning to physical-safety metaphors, but it's just too similar to what a self-defense instructor friend of mine used to tell me. He couldn't provide specific actions for a given attack,

such as “When he grabs your arm kick him in the groin” (a rather ineffectual way of deterring a determined attacker incidentally), because attacks aren't scripted. Defense needs to be based on principles, such as “against a stronger attacker, your safest position is in close”, rather than given techniques.

In both situations the most important work is up to the company or person seeking security and defense. A secure system is the result of an intimate knowledge of individual security requirements and limitations. Consultants are valuable for providing technical know-how and pointing out possibilities, but your network security is ultimately work that must be done by you.

Rob Beck's (another @staker) article in this month's feature section is a good example. He provides a great little application for fingerprint evasion, but the level of anonymity (and even whether anonymity is high on one's security priority list) is up to the user, as Rob points out.

In addition to the usual Paranoid Penguin and security feature articles, this issue's Kernel Korner, Focus on Software and Take Command are also secure-centric. In fact, we ended up with so many HOWTO security articles that a number of them couldn't be squeezed into the print magazine and were relegated to the infinite space of our web site—see the Strictly On-Line section of the contents page for titles.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

Best of Technical Support

Various

Issue #89, September 2001

Our experts answer your questions.

The Dreaded “LI” Problem

I put a second IDE hard drive in my PC and installed Mandrake Linux 8.0 on it. Now when I reboot, I have the dreaded “LI” problem, i.e., LILO displays only the first two letters of the LILO prompt. I believe the source of my problem is that I installed the second hard drive as slave on the secondary IDE. My boot partition (mounted as /) is hdd6.

—Frederic Mora, fmora@attglobal.net

Getting “LI” from LILO usually means that the first-stage boot loader was able to load the second-stage boot loader but has failed to execute it. This can either be caused by a disk's physical configuration inconsistency or because you moved /boot/boot.b without running the map installer. In practical terms, you indeed need to reconfigure /etc/lilo.conf to describe where (which disk) each partition is on your system. So, you need to reboot your system from a rescue disk and get into a root prompt so you can edit lilo.conf and point each element to where it is physically located (in accordance to the way your disks are identified: /dev/hdaX, /dev/hdbY, /dev/hddZ, etc.). After editing /etc/lilo.conf, run **/sbin/lilo -v** to write all to the disks and reboot. Look at the page www.fan.nb.ca/~aa126/troubleshoot-LILO.html for additional information on LILO.

—Felipe E. Barousse Boué, fbarousse@piensa.com

wu-ftp Won't Let Users in

I'm having some problems with logging on to the FTP server in my company. I've installed wu-ftpd and anonyftp to the server. The guest account has been

created, but all the users are not able to log on to the FTP server. I'm sure that the password is correct.

—Alan Lim, alan_lim@astro.com.my

Check the log files in /var/log to see if they are telling you anything. You can use the -d flag in conjunction with wu-ftpd to increase the amount of logging. If you compiled the FTP daemon yourself, check to see whether it is having problems with PAM or shadow passwords. It may not be sensing that this is required during the compile process. Please note that, in any case, this is a huge security risk. **wu-ftpd** is not designed to execute the chroot() function for normal user logins, meaning normal (nonanonymous) users will be able to access the entire system once they log in. You might want to consider fixing both issues at the same time by installing a more secure FTP daemon such as ProFTPD or NcFTPD and configuring them to chroot() to the user's home directory.

—Chad Robinson, crobinson@rfgonline.com

Alan, please make sure that the shell account used with those users is listed in the wu-ftp configuration. Otherwise it will deny access even if the password is correct.

—Mario Neto, mneto@argo.com.br

I Have No Source and I Must make

I have installed or attempted to install almost every distribution available. The best install I have found is Mandrake 7.0, 7.1 and 7.2. The problem I have is I cannot get source code to install when I do **configure**. The system either gives me a syntax error or some file is missing. I've read all the read files, I have several books and I still can't find out what's wrong.

—Bill York, bill_york@pipeline.com

You should be able to install source files (such as the Linux kernel source) with RPM; just mount the source CD, **cd** to the directory containing the source RPMs, and use

```
rpm -i kernel-source-file.rpm
```

That should do it. Installing with RPM (in the case of Mandrake) is fairly easy, and it also takes care of most dependencies on other files. If you still get an error, then you probably need to install some other stuff before the actual file you are trying to install.

—Felipe E. Barousse Boué, fbarousse@piensa.com

Ignoring BOOTP Requests

I recently set up a DHCP server, and when I monitor the port I am seeing BOOTP requests that come from an old VAX system that gets its boot information from a machine on the other side of our router. Is there something I can turn off on my Linux box or configure to ignore those requests?

—Pat Derosa, pderosa@ap.org

Are the BOOTP requests actually causing problems or are they merely an annoyance in your log files? The ISC DHCP daemon allows you to use the deny bootp option to ignore BOOTP clients, but that may not stop the server from logging the request. In that case, you may have limited options. You may be stuck with the message unless you are comfortable locating the line in the source code, commenting it out and recompiling the daemon.

—Chad Robinson, crobinson@rfgonline.com

If you don't have "allow unknown-clients", your DHCP server will not serve requests to machines that aren't explicitly listed by MAC address. Also, if you omit dynamic-bootp, your DHCP server will not serve bootp requests.

—Marc Merlin, marc_bts@valinux.com

Adding Services from Kickstart

In writing the post installation part of my Red Hat 7.1 Kickstart file, I would like to add services such as ypserv and autofs automatically. How can I do it?

—Sowmya, sowms@yahoo.com

chkconfig would be the way to do it. Do a **chkconfig --help** and see all the options you can. For instance, add nfs to start automatically at run level 3 with

```
chkconfig --level 3 nfs on
```

—Felipe E. Barousse Boué, fbarousse@piensa.com

Slow Mail Server

We are experiencing a long delay in resolving the request at the server when checking for mail. At times it is instant (seldom) and at others it times out (taking more than four minutes). We have tried with different setups of "hosts, DNS" and "DNS, hosts", assuming it was trying to resolve the querying address.

When a dial-up connection is made they are assigned an IP address and DNS server.

The Internet (Squid) has no problems and works perfectly. It also seems that once a connection is made checking for mail, the next request is instant.

—Kevin, kevin@atom.co.za

You could use **host mailserver** and check the delay to see if it is DNS. You could use **tcpdump** and watch traffic to and from your mail server to see where the delay is (assuming it is not DNS).

—Christopher Wingert, cwingert@qualcomm.com

You should run **tcpdump**, or if you have it, **ethereal**, and snoop your connection. This will give you an idea if the DNS delay is from your side or possibly on the mail server's side.

—Marc Merlin, marc_bts@valinux.com

Need to Write to Windows Partitions

Although I have my three Windows partitions mounted in Linux, I only have write-access when logged on as root. I need to have write-access as myself so that I can run VMware. I have tried to change the permissions for these partitions using **chmod** while logged on as root, but the permissions don't change.

—Bill Freeto, wfreeto@earthlink.net

Use the uid and gid mount options (you also want "quiet" typically). A sample fstab entry for a vfat partition would look like this:

```
/dev/sda3    /drv/c      vfat
user,umask=002,uid=500,gid=500,quiet,low
```

More info can be found in the mount man page.

—Marc Merlin, marc_bts@valinux.com

That's Enough Disk Space for You

Is there any way that I could limit the size of a directory? I want to start a web-hosting server and limit some users to 100MB. How do I go about doing this?

—Jason Sidabras, sidabra@msoe.edu

Take a look into the quota package for Linux. Part of this application resides in the kernel and must be compiled in or installed as a module. The other part is a user-space program that handles the actual control and notification aspects.

—Chad Robinson, crobinson@rfgonline.com

Look at the Quota mini-HOWTO: www.linuxdoc.org/HOWTO/mini/Quota.html.

—Marc Merlin, marc_bts@valinux.com

2.4.2 Panics!

I tried to upgrade my kernel to 2.4.2. When I rebooted I got a kernel panic. It said:

```
root fs not mounted
cannot open root device "301" of 03:01
Please append a correct root = "boot option"
kernel panic vfs:
unable to mount root file system on 03:01.
```

I used the same device as my other kernel, /dev/hda1, and I did a **rdev** to make sure that is right. I also went back into **make xconfig** and made sure that ext2 was compiled in and not a module.

—Michael Diaczyk, mdiaczyk@tampabay.rr.com

If you didn't somehow mess up the partition table or the actual disk data, you should be able to boot by passing a parameter to LILO at boot time. At the LILO: prompt, try typing

```
linux root=/dev/hda1
```

Once this works, edit your /etc/fstab to ensure that the entry for your root (i.e., /) partition is correct. Also ensure that the root= line in your /etc/lilo.conf points to the right partition.

—Scott Maxwell, maxwell@ScottMaxwell.org

Bad Module, Bad, Bad!

Is it possible for an errant module to mess up /proc files so that nobody can look at them without causing an oops? If the module forgets to free an interrupt when it is unloaded, then /proc/interrupts is gone. If it forgets to release I/O space, then /proc/ioports is defunct. If it calls unregister_chrdev with the module name misspelled (nobody would do that, right?), that destroys /proc/devices. When this sort of thing has happened, can the system be rescued or is rebooting the only option?

—Bill McConnaughey, mcconnau@biochem.wustl.edu

It's possible for an errant module to do anything—it's running as part of your kernel, after all. If I saw a problem like the one you're reporting, I'd reboot immediately. It's theoretically possible to fix the problem without rebooting by writing and installing another module that undoes the damage caused by the first one, but this is impossible in practice unless you happen to know exactly what went wrong—and it's not necessarily easy even then.

—Scott Maxwell, maxwell@ScottMaxwell.org

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

[Advanced search](#)

New Products

Heather Mead

Issue #89, September 2001

PBS Pro 5.1, ClusterWorX 2.0, Javlin, and more.

PBS Pro 5.1

Version 5.1 of Veridian's PBS Pro is now available in the professional edition for workload management and batch queuing on clusters. Job-tracking capabilities and a common user interface allow PBS Pro to be used on all UNIX platforms. Features in version 5.1 include dynamic load balancing, cross-system scheduling, job interdependency and chaining, pre-emptive job scheduling, management for batch and interactive work, and graphical and command-line interfaces.

Contact: Veridian Systems, PBS Products Dept., 2672 Bayshore Parkway, Suite 810, Mountain View, California 94043, 877-905-4PBS (US toll-free), sales@pbspro.com, www.pbspro.com.

ClusterWorX 2.0

ClusterWorX 2.0 is Linux NetworX's management software that allows administrators to control the cluster as a single system. It allows users to monitor a cluster system remotely down to the individual node. Customizable and extendable, ClusterWorX also employs an event engine that can be set up for a variety of variables. Features of ClusterWorX include visually based monitoring, plugin support, three-tiered application, integrated disk cloning and image management, and node power monitoring.

Contact: Linux NetworX, 8689 South 700 West, Sandy, Utah 84070, 801-562-1010, info@linuxnetworx.com, www.linuxnetworx.com.

Javlin

Javlin, Object Design's middle-tier enterprise JavaBean data cache manager, has been extended to Linux i386 environments, supporting version 2.4 of the kernel. Javlin stores and delivers data to application servers using distributed, persistent caches. Javlin also allows dynamic addition of categories and attributes while the system is in production. In addition, it provides a Java dynamic data framework.

Contact: Object Design, 25 Mall Road, Burlington, Massachusetts 01803, 800-424-9797 (US toll-free), info-objectdesign@objectdesign.com, www.objectdesign.com.

Pervasive.SQL 2000i

Pervasive Software, Inc. released Pervasive.SQL 2000i, embedded database software for building applications that can migrate to the Web and across platforms without code changes. The multiplatform database engine supports version 2.2 and higher of the kernel. 2000i provides backward compatibility with existing API-based applications on Linux, NetWare and NT platforms. In addition, it offers server/requester version independence and integration without configuration changes. Pervasive.SQL supports the ODBC, OLE DB 2.5 and JDBC 2.0 standards, as well as Java, Perl, PHP and other languages.

Contact:Pervasive Software, Inc., 12365 Riata Trace Pkwy., Bldg. II, Austin, Texas 78727, 800-287-4383 (US toll-free), info@pervasive.com, www.pervasive.com.

Replicator

everStor, Inc. announced the availability of Replicator, its management, replication and archival software. Replicator centralizes control and management for replicating specific disks, directories or files from any server or workstation within an enterprise over a LAN, WAN or the Internet. As it provides on-line data availability, Replicator can be used as part of a disaster recovery program. Residing on the server or network appliance, Replicator requires no installation of client-side software. File transfer occurs over a TCP/IP connection.

Contact: everStor, Inc., 1240 N. Lakeview Avenue, Suite 150, Anaheim, California 92807, 714-970-7511, sales@everstor.com, www.everstor.com.

Sistina GFS 4.1.1

Sistina's GFS is a clustered filesystem that allows multiple servers on a SAN to have read/write access to a single filesystem on shared SAN devices. New

features of version 4.1.1 include GFS kernel patches for Linux 2.4.5, a new I/O fencing method (apc_ms network power switch) and updated initialization scripts. GFS 4.1.1 is released under the GNU GPL and can be downloaded at www.sistina.com/gfs/software/index.html.

Contact: Sistina Software, Inc., 1313 Fifth Street SE, Minneapolis, Minnesota 55414, 612-638-0507, www.sistina.com.

MandrakeSecurity

MandrakeSoft, publishers of Linux-Mandrake, announced the release of the Single Network Firewall, the first in its new line of security products branded MandrakeSecurity. Designed specifically for use by small- and medium-sized businesses and SOHOs, it includes an easy-to-use remote web interface, a secured connection for the network, and filtering, intrusion detection and proxy features. Released under the GNU GPL, it can be downloaded from <http://www.linux-mandrake.com/>.

Contact: MandrakeSoft, Inc., 2400 N. Lincoln Avenue, Altadena, California 91001, 626-296-6290, www.mandrakesoft.com.

[Archive Index](#) [Issue Table of Contents](#)

[Advanced search](#)

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.