Information Security on Linux

Paranoia for Beginners

If an attacker gains direct access to a computer, any data stored there is up for grabs, even if it means physically removing the hard disk.

BY BÖJRN GANSLANDT

No matter how well your firewall is configured, the only way to protect yourself is to use encrypting software, and that is what we will be discussing in this article.

Encryption

If you have a large number of files to **encrypt**, it does not make sense to do so on a file for file basis; instead you should opt for an encrypting file system. Fortunately, this does not require a new partition, because you can use the loopback device to mount normal files as block devices (such as hard disk partitions) and encrypt them on the fly during the process.

Unfortunately, the standard loop driver only supports an **XOR** algorithm due to export restrictions on cryptography that apply in some countries. This may protect your data from over inquisitive siblings, but it will not stand up to a serious attack. So you will need to apply a **patch** to add additional encryption algorithms to the driver.

The International Kernel Patch, alias CryptoAPI [1], which adds a variety of algorithms to the kernel, thus allowing encrypting file systems to be used, has traditionally provided the best solution. However, development of CryptoAPI has made very slow progress since the release of Linux 2.4, and the installation is comparatively complicated. The gap caused by this lack of progress was quickly filled by Loop-AES [2] – as the name would suggest, this tool is restricted to AES encryption.

AES (Advanced Encryption Standard) was originally known as Rijndael and was chosen as the successor for DES after months of testing by the National Institute of Standards and Technology (NIST) and many other cryptoanalyists, so it is safe to assume that the algorithm is robust.

Before installing Loop-AES you need to expand, configure and compile the kernel source code below */usr/src/linux*, as Loop-AES requires access to the unpatched version of *loop.c* and several kernel settings. When configuring the kernel ensure that the modules *Kernel Module Loader* are enabled, and that *Loopback device support* under *Block Devices* is disabled.

After installing the new kernel and its modules, Loop-AES can be extracted to a directory and compiled by typing *make*. This step will not affect the current kernel source code, as Loop-AES simply patches a local copy of *loop.c* (or accesses a previously patched version) and then installs this as a module. AES support for the kernel is insufficient in itself, as the *mount*, *losetup* and *swapon* programs also need modifying, although the latter is only needed if you intend to use an encrypting swap partition. All of these programs are components of Util Linux, which is available from [3].

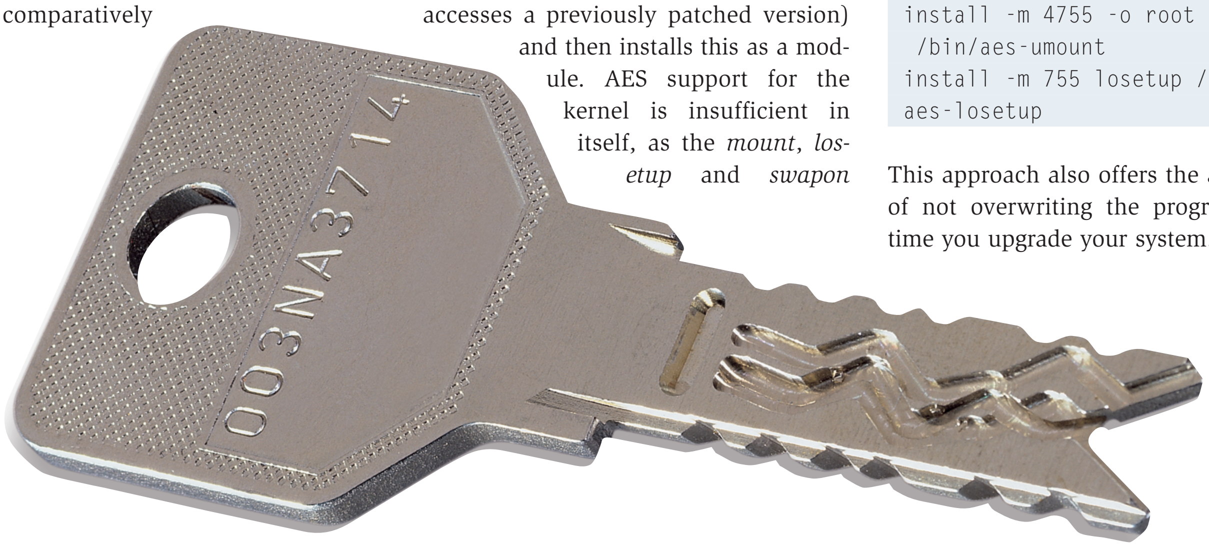
The download version of the archive must match the Loop-AES patch to compile without errors. If the patches are in a different directory, you will additionally need to modify the path variable:

```
patch -p1 < ../util-↩
linux-2.11y.diff
export CFLAGS=-O2
./configure
make SUBDIRS="lib mount"
```

These steps should place the required tools in the *mount* subdirectory. You should avoid installing the programs in */bin* or */sbin*, which would overwrite the original versions, as this may cause conflict with other system components. It is safer to use an alternative name when installing the programs:

```
install -m 4755 -o root mount↩
 /bin/aes-mount
install -m 4755 -o root umount↩
 /bin/aes-umount
install -m 755 losetup /sbin/↩
aes-losetup
```

This approach also offers the advantage of not overwriting the programs each time you upgrade your system. The only



disadvantage is the fact that the test routine *make tests* in the Loop-AES directory does not know about the alternative names, and thus issues an error message.

Now it is finally time to create the encrypting file system. To do so, first create a new file that will contain the encrypting file system. You can alternatively use a hard disk partition – leave out the following step in this case. As you cannot increase the file size later, ensure that the file you create is large enough:

```
dd if=/dev/urandom ⮐
of=./secret bs=1024k count=20
```

This command redirects random data from */dev/urandom* to the file *./secret*; the file size is the product of *bs* and *count* – that is 20 MB in our example. */dev/urandom* uses various internal system events to generate random data exactly like */dev/random*. Where */dev /random* will freeze if insufficient seed data is available, a pseudo-random numerical generator will continue to produce output for */dev/urandom*, and this is perfectly okay for the task in hand.

The next step involves setting up the loopback device */dev/loop1*. If you want to encrypt a partition, supply the device name (e.g. */dev/hdb7*) instead of a file name in this step. Of course, this will destroy any data stored on the partition:

```
aes-losetup -e AES128 -T /dev/U
loop1 ./secret
```

losetup should now prompt you for a passphrase with at least 20 characters. If you use the 192 or 256 variants instead of AES128, the minimum length of the passphrase will be 32 or 43 characters. After the system has accepted your password, you can create a file system on the device you have created and then disable the device:

```
mkfs -t ext2 /dev/loop1⮐
aes-losetup -d /dev/loop1
```

Of course, you can opt for other file systems, but *ext2* is your best bet in this case. Finally, you will need to add an entry to */etc/fstab* to simplify mounting the device in future – of course, there are no restrictions on the **mountpoint**:

```
/directory/secret /mountpointU
 ext2 defaults,noauto,loop=U
/dev/loop1,encryption=AES128 0 0
```

Provided you have the passphrase, you should now be able to access the file system via *aes-mount /mountpoint*.

Wipe Out

Before you move your confidential data to the encrypting file system, you might like to consider how your non-encrypted data can be effectively wiped. It is by no means sufficient to delete this data, as deleted files are simply cleared for overwriting. Although the data no longer shows up in the file system, it is still on your hard disk and can be restored with little effort.

Data that has been overwritten is slightly trickier, but high resolution microscopes should be able to reveal data despite multiple overwrites, as the read heads only provide a certain degree of accuracy and traces of previous data survive at the edges of the current magnetically coded bits.

Erasing data beyond the means of any recovery procedure involves overwriting the data with random noise and specific patterns tailored to reflect various data encoding techniques. Wipe [4] and Secure Delete [5] are specifically recommended for this purpose; note that this refers to the Wipe version by Berke Durak and not to the identically named program by Tom Vier [6].

srm, the Secure Delete equivalent to *rm* overwrites each file 38 times by default, whereas Wipe makes do with a mere 34 times; of course it is debatable whether the four additional operations make a big difference. Both programs additionally rename the file in order to destroy the file name and are capable of processing directories recursively. A quick (but unsafe) mode is available for both programs; use the *-q* parameter for Wipe and *-f* for *srm*. The tools that Secure Delete provides in addition to *srm* are the major difference between the two tools.

For example, *sfill* can be used to safely erase the free space on a partition, and *swap* will also delete the swap content, which could otherwise reveal data swapped out of main memory. Extremely paranoid user can additionally launch *smem* to overwrite the contents of the main memory.

Hideaway

Although cryptographic tools are quite capable of protecting the contents of a message from inquisitive third parties, an encrypted file or email message will tend to show up in a mass of clear text messages – and this in itself might attract the attention of an intruder.

Steganography (the art of hidden writing) provides a solution to this issue by hiding information in a harmless host medium such as an image without any recognizable manipulation.

JPEGs and other image formats are typical hosts as they are inconspicuous and large enough to transport confidential data. The simplest, and for this reason probably the most common tech-

GLOSSARY

Encryption: *An algorithm that typically uses a key is applied to the clear text to transform it into an encrypted message.*

XOR: *XOR (exclusive or, ⊕) is an operation that returns 0 when both bits it is applied to are identical. The result is 1 for different values. XOR encoding simply adds the clear text and the key bitwise. The same procedure is repeated to decrypt the message, as the key cancels itself out ($K \oplus S \oplus S = K$). This type of encryption is only safe if the key is not repeated; that is where the key is the same length as the message itself (see* **One time pad***).*

Patch/Kernel patch: *A patch file contains instructions on modifications to one or multiple files. This saves download time, as you only need to apply the patch file for the new program version to the existing source code, instead of having to download the new source code.*

PRNG: *A Pseudo Random Number Generator is an algorithm that outputs extremely long, seemingly random data sequences. If the PRNG is used in a cryptographic context, predictable output must be avoided as the entire cryptographic infrastructure is otherwise vulnerable. Open source algorithms avoid this by using a random seed that needs to be kept secret, just like a cryptographic key, and often comprises user dependent events.*

Mountpoint: *A mountpoint is the point where a file system is inserted. After mounting, the content of the file system is displayed as part of the directory chosen as the mount point hiding any data in this directory.*

nique to embed data in uncompressed image formats, such as BMP, is to alter the least significant bits of the RGB values (see Insert 1) – after all, this does provide three bits of storage per pixel.

Only the least significant bits are manipulated, as they have the least impact on color values, and the effect on the image is thus restricted to more or less invisible color nuances.

JPEGs images are a more complicated issue, as the image is not simply stored pixel-wise, but translated into frequency coefficients by so-called **discrete cosine transformation**.

The coefficients are subsequently quantised, which means that high-frequency coefficients (details that the eye can hardly detect) are rounded to zero. Some image information is lost during this process, and this is why JPEG is referred to as an image loss compression method.

As the image is then additionally compressed (without any further loss) it is more difficult to manipulate the actual bytes than it would be with a BMP, for example. Instead, the least significant bits of the individual coefficients are overwritten before the loss-free compression of the image begins.

In both cryptography and steganography there is constant competition between new algorithms and analytical processes. Thus, the above mentioned method can be broken both by visual attacks and a statistical method. The method relies on recognizing atypical hue value distribution patterns in manipulated images.

A program called OutGuess [7] provides one possible solution to this issue, as it avoids changing the typical frequency coefficient distribution patterns and is thus resistant to analytical methods. In practical terms, this means that for every frequency coefficient manipulated within an image, a matching coefficient is manipulated in exactly the opposite direction.

So, if a value of 2 is replaced by a 3 at one position, OutGuess will convert a 3 to a 2 at another position. The Java tool F5 [8] also resists statistical attacks by decrementing the coefficient values instead of simply overwriting individual bits; the JPHS tool [9], which we will not be looking into in this article, is equally resistant.

Although they use different mathematical methods, all of these programs are used in a similar way. You need a host image, the file with the data to be hidden, and a password that is used to additionally encipher the data. The additional cipher is required to prevent the hidden file from being revealed simply by launching the program, depending on the strength of the algorithm.

You should additionally be aware of the fact, that an image cannot be used more than once, and that the original should not be available on the Internet or from any other public source, as the steganographic message could be extracted by simply comparing the image with the original. After selecting a suitable image, you can use the following commands to embed the hidden information:

```
outguess -k "Bigger is Better"➥
 -d crypto.txt my.jpg my2.jpg
```

You need a Java Runtime Environment to run the F5 tool. The syntax is as follows:

```
java -mx40M -classpath $CLA➥
SSPATH:/usr/local/f5 Embed ➥
-e crypto.txt -p "Bigger is ➥
Better" my.jpg my2.jpg
```

The *$CLASSPATH* environment variable contains the path to the classes of the Java Runtime Environment and */usr/local/f5* is the directory where the F5 tool resides. If you use JRE 1.4 by Sun or Blackdown, the command should work without the *$CLASSPATH:* syntax. The secret message is extracted using the reverse syntax:

```
outguess -k "Bigger is Better"➥
 -r my2.jpg crypto2.txt
```

Or with the F5 tool:

```
java -mx40M -classpath $CLASS➥
PATH:/usr/local/f5 Extract -e ➥
crypto2.txt -p "Bigger is ➥
Better" my2.jpg
```

Unfortunately, new mathematical methods have been developed to break the algorithms used by F5, OutGuess and JPHS. The program authors will of course respond to the challenge – but it is safe to assume that the capacity for hiding information will drop again in the next generation of software. So it makes sense to use an MP3 file as a host medium. Even if the embedding capacity drops to less than one percent, an MP3 should still offer plenty of space. A program called MP3Stego leverages this potential and embeds messages while encoding MP3 files [10]. The following commands are used to hide or extract messages – note that the WAV file must be 16 bit format:

GLOSSARY

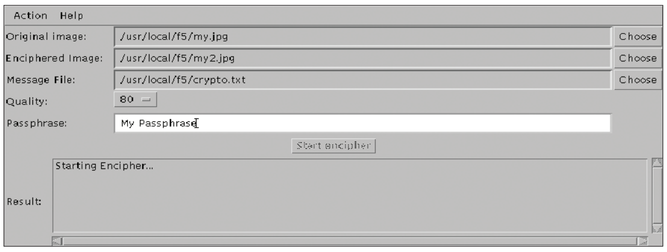
Discrete Cosine Transformation: *Discrete Cosine Transformation (DCT), which is closely related to Fourier transformation, is applied to an image block measuring 8x8 pixels during JPEG encoding and transforms the three dimensional topology (the third dimension is derived from the pixel values) of the block to a discrete frequency amplitude assignment. The frequency of the individual frequency coefficients is used as the scale for modifications to the image; thus a uniform surface will be represented by a low frequency, whereas high-resolution details will be represented by higher frequencies.*

History/.bash_history: *The Bash (and most other shells) store the commands typed during a session in a history file (which defaults to ~/.bash_history). You can use the unset HISTFILE syntax to delete the environment variable that points to the history file and thus make it impossible to store any commands.*

One time pad: *One time pad involves encrypting a message with an absolutely random key of the same length as the original clear text. Traditionally, a modulo 26 (that is the remainder after dividing the sum by 26) addition was performed for the each letter of the message and the key. Today, most people tend to XOR binary messages and keys. Provided that the key really is random, and is only used once, there is no way to break this algorithm. The disadvantage is that you need a large amount of random data, and that the key is just as long (and thus in many cases equally as difficult to keep a secret) as the message itself.*

Keyboard logger: *A keyboard logger is a program or device that grabs keyboard input, thus revealing passwords and other confidential data.*

PGP/GPG Mantra: *A mantra is a password used by PGP and GPG to encrypt the private key and protect it from undesired access.*



```
Original image:   /usr/local/f5/my.jpg
Enciphered Image: /usr/local/f5/my2.jpg
Message File:     /usr/local/f5/crypto.txt
Quality:          80
Passphrase:       My Passphrase
Result:           Starting Encipher...
```

Figure 1: F5 also offers a GUI frontend

```
encode_ix86 -E crypto.txt ↩
my.wav my.mp3
decode_ix86 -X my.mp3 ↩
my2.wav crypto2.txt
```

MP3Stego insists on playing the WAV file (or what's left of it) in addition to outputting the hidden message, but you can prevent this "feature" by specifying */dev/null* instead of *my2.wav*

One particularly convincing feature that MP3Stego offers is the fact that you are not required to supply a passphrase when launching the program.

Instead, you are prompted for the passphrase after launching the tool. Unfortunately, all the other tools do not offer this feature – so the passphrase ends up in your shell **history**. If you want to keep your password a secret, you might like to disable your (Bash) history by typing *unset HISTFILE* for the remainder of the current session.

Box 1: Least significant bits

Many steganographic algorithms manipulate only the least significant bit. To understand how this works, you must be aware of how numbers are represented in a binary context. Let's use a pixel in a 24 bit BMP file as an example: the hue values for this pixel are defined by the ratios of red, green, and blue (RGB). Our BMP file uses one byte (8 bits) for each hue value, that is 24 bits per pixel.

BMP files store hue values in inverse order. The binary representation is thus as follows:

	Blue	Green	Red
Decimal	198	113	47
Binary	11000110	01110001	00101111

If the bits 101 are now embedded, the least significant bits in the three color bytes are modified as follows:

	Blue	Green	Red
Decimal	199	112	47
Binary	11000111	01110000	00101111

Expressed as a decimal, modifying these bits will mean a maximum deviation of 2

that is 0, +/- 1. Thus the least significant bit has the smallest possible affect on the numerical value, and the hue; in fact, the human eye should not be able to detect any difference from the original.

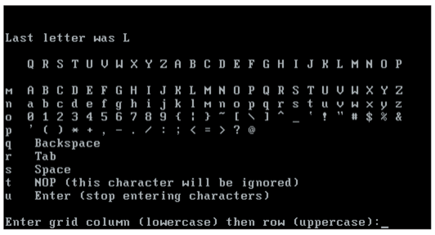
Advanced Paranoia

When dealing with encryption, you should be aware that neither AES, nor any other cryptographic algorithm (with the exception of the extremely impractical **one time pad**) can be mathematically proven to be secure, although they have proved resistant to attacks launched by the international cryptoanalysis community.

Additionally, it is by no means sufficient to regard the algorithm as an isolated occurrence – often, a completely different area of the system will prove to be the weakest link. Thus, a **keyboard logger** or a cleverly hidden camera can reveal any data entered by a user, including their **PGP/GPG mantra**, and circumvent any security measures.

Electromagnetic emission is another weak point of typical computer systems, and almost any PC component (especially the display) will produce it. A program called Tempest for Eliza [11] shows how easy it is to receive emissions, even using the most simple means, by creating special patterns on screen that can be picked up by radio receivers.

Of course, it is possible to insulate your computer room, but signals are transmitted by any cables to which the computer attaches and this makes insulation extremely difficult. Your best bet is to use a laptop in an insulated room, and if possible, to run Tinfoil Hat Linux [12] on it. This distribution fits on a single floppy and comprises both GPG and (Tom Vier's) Wipe. *gpggrid* provides protection against keyboard loggers by allowing you to choose a mantra letter by letter from a randomly generated matrix.



```
Last letter was L

   Q R S T U V W X Y Z A B C D E F G H I J K L M N O P

m  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
n  a b c d e f g h i j k l m n o p q r s t u v w x y z
o  0 1 2 3 4 5 6 7 8 9 { | } ~ [ \ ] ^ _ ' ! " # $ % &
p  ' ( ) * + , - . / : ; < = > ? @
q   Backspace
r   Tab
s   Space
t   NOP (this character will be ignored)
u   Enter (stop entering characters)

Enter grid column (lowercase) then row (uppercase):_
```

Figure 2: gpggrid provides – admittedly roundabout – protection from keyboard loggers

Additionally, the "Paran0id" option in the Tinfoil menu activates an extremely low contrast color mode that should make eavesdroppers life difficult (or your optician happy, as the case may be). With this option GPG continually encrypts files in the background, to distract you from your own GPG instance.

Another tool that is useful in situations where you cannot trust your own screen is *morseblink*, which can be used to send messages by morse code via the keyboard LEDs. In "Paran0id" mode *morseblink* is used to transmit random morse code and thus overlay any emissions the keyboard might produce. In standard mode you can use the following command to morse code a file:

```
cat text | morseblink
```

Finally, the Tinfoil Hat README [13] recommends wearing a hat made of aluminium foil to protect your own thoughts from both external influences and reading – although one might be tempted to question the seriousness of this recommendation. ■

INFO

[1] *http://www.kerneli.org/*

[2] *http://loop-aes.sourceforge.net/*

[3] *http://www.kernel.org/pub/linux/utils/util-linux/*

[4] *http://gsu.linux.org.tr/wipe/*

[5] *http://freshmeat.net/projects/securedelete/*

[6] *http://wipe.sourceforge.net/*

[7] *http://www.outguess.org/*

[8] *http://wwwrn.inf.tu-dresden.de/~westfeld/f5.html*

[9] *http://linux01.gwdg.de/~alatham/stego.html*

[10] *http://www.mirrors.wiretapped.net/security/steganography/mp3stego/*

[11] *http://www.erikyyy.de/tempest/*

[12] *http://tinfoilhat.shmoo.com/*

[13] *http://www.zapatopi.net/afdb.html*