Biometrics and Linux

Alexander Reeder
alexr@valinux.co.jp
VA Linux Systems Japan, K.K.

January 2003

Introduction

Whether it is a corporation, airport, or government agency, biometrics are becoming an extremely popular way of identifying and classifying people. There are many devices which employ biometric methods, such as fingerprint scanners, voice recognition, and retinal scanners. While a person's retina is the most statistically unique part of one's body [1], due to the retinal scanners inhibitory price this paper focuses on fingerprint scanners.

Biometrics and Linux are divided into three sections. The first section delves into integrating a biometric form of identity verification with Linux's authentication system. Next we cover filtering out noise from a fingerprint scan for the verification algorithm. Finally, the paper is closed with a brief discussion of how fingerprint matching is actually performed.

Section 1: The Big Picture

There is a multitude of companies producing fingerprint scanners: BioEnable Technologies Private Limited [2], DigitalPersona, Inc. [3], and Biometrika srl [4], are good examples of major players in the market. The best way to find available scanner manufacturers is via Google [5]. Manufacturers often provide authentication software with their products, but also sell SDK's to allow software vendors to develop alternative solutions. A good example is Neurotechnologija, Ltd. [6] whose software works on any of the above scanners. Neurotechnologija is also the only vendor which provides source code and Linux drivers (for both the device and its authentication software). However it is closed source and not free. IBM in collaboration with Consumer Direct Link, Inc. also recently released the worlds first biometric authentication based PDA [7]. There are more vendors of recognition software and fingerprint scanners than listed above, however the focus of this paper is on implementation. Readers are encouraged to do their own research on available solutions.

Figure 1: DigitalPersona's U.are.U scanner [8]

For implementation DigitalPersona's U.are.U was used (Figure 1, above). It met the requirements: a generic scanner that has a decent scanning quality of 500 DPI and a USB interface. The first task at hand was writing a driver for the U.are.U scanner. The retailer, DigitalPersona does not provide online documentation for the device. After contacting them, declined to release it. One could purchase the necessary technical documentation from DigitalPersona to develop a driver, but the price is prohibitive.

At this point I would like to offer some tips to those attempting to procure documentation from hardware vendors for the purpose of writing a driver. If just asking for the documentation fails, try different approaches. You might contact the company as a representative for an educational institution who is interested in developing a driver for study purposes. If you are interested in commercially deploying the product, say so and request the technical specifications of the product for evaluation. Using such methods Cameron Murrell successfully obtained the specs for the parallel scanner we worked on, circa 1996 [9], and which I later replaced with the U.are.U. Be creative, consistent and persistent.

Without documentation, but assuming the scanner worked on simple principles, I utilized a USB sniffer [10] to 'view' the interaction between a workstation and the U.are.U. Using that data I was able to write a simple driver to provide data to the authentication (PAM) module (Figure 2, below).

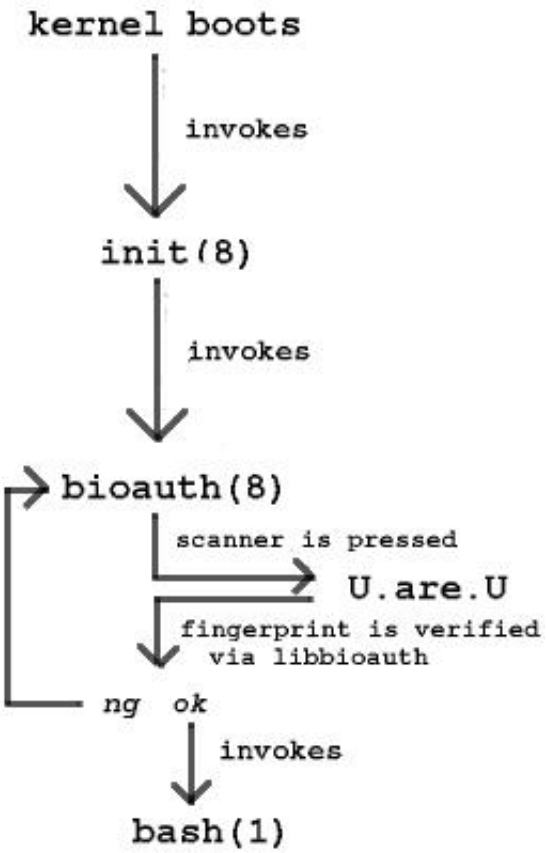

Figure 2: General flow, from boot to authentication

Normally init(8) is invoked as the last step of the kernel boot process. On a typical linux distribution, say Debian, /sbin/getty is launched for 6 virtual terminals (tty1-6). After typing in a username, getty(8) invokes /sbin/login, which reads the password and then authenticates the user via a PAM module

which typically uses the /etc/passwd and /etc/shadow files. The approach outlined above authenticates through a new program, /sbin/bioauth. bioauth(8) registers itself with the U.are.U USB module. If the U.are.U scanner is not present bioauth(8) will exec(3) getty(8) for normal authentication. This feature can be turned off via the command line. Normally a /etc/inittab entry for bioauth(8) looks like this:

1:2345:respawn:/sbin/bioauth tty1 none

The first option tells bioauth(8) which tty to attach to, and the second option is the program that should be exec(3)'ed when the U.are.U fails to initialize (due to its lack of presence, etc.). 'none' could be replaced by '/sbin/getty' or '/sbin/mingetty' (any option following the replacement program is also executed as an option). If you are using the U.are.U scanner (or any solution of its type) you should remove the other virtual terminal entries from /etc/inittab leaving one entry for bioauth that has 'none' for its secondary authentication program. Remember this is risky if you are working with flaky authentication software or hardware because it can lock you out of your system.

After bioauth(8) has received the image data from the U.are.U, it relies on a library, libbioauth. libbioauth contains the functions necessary to filter and authenticate the scanned image. Proper authentication will involve three calls to the library.

1. Convert the raw data into an image of size x,y.
2. Filter the converted image to reduce noise and make it easier to verify (See Section 2).
3. Authenticate the image, returning either success or failure (See Section 3).

There are three other required works in progress, useraddbio and userdelbio to add and remove users, and userviewbio to view registered users data. At the time this paper was composed I resorted to manually inserting and removing fingerprint data.

Section 2: Gray, grays, and more gray

From the U.are.U scanner we receive data which can be transposed into a gray-scale bitmap image. Being a gray-scale bitmap the range of grays is from 0 (black) to 255 (white). The U.are.U has a DPI of 500 and an image capture area of 13 x 18 mm. The quality of the image received largely depends on the caliber of the technology behind the actual scanning. Scanners made for scanning fingerprints are designed to reduce the effort needed for filtering the image to obtain useful data. The U.are.U is unremarkable, physical magnification of its capture area being its only differentiating characteristic from off-the-shelf scanners. Some interesting non-conventional work in this area is being done by the Innovative System Design Group at the University of Bologna, Italy [11].

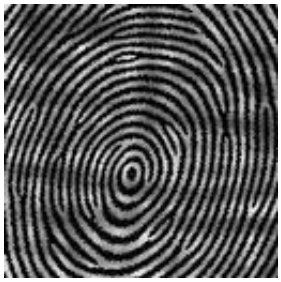

Figure 3: An unfiltered fingerprint

Figure 3, above is an example of an unfiltered mock image from the U.are.U [12]. Before passing this image on to the authentication library it needs to be filtered. Filtering can be accomplished by turning the bitmaps dark grays into black (ridges) and the mid-light grays (valleys) into white. The process for

doing so is detailed below.

1. If white (or light gray) is present, it is determined to be outside of the fingerprints boundary, and is subsequently removed (by turning the area to white).
2. The number of occurrences of each gray tone in the fingerprint is recorded. The most common darkest gray is considered to make up the backbone of the ridges. The most common lighter gray is considered to make up the valleys.
3. The difference between the ridges and valleys is calculated and halved. Any of the grays greater than black (0), and less than the most common darker gray (ridge) value plus half of the difference, are turned black. Any values greater than the lighter gray (valley), and greater than the lighter gray plus half the difference, are turned white.

l = most common darkest gray
d = most common lightest gray
x = (l - d) / 2
Ridges = 0 <= l <= (l+x)
Valleys = (l+x) < d <= 255

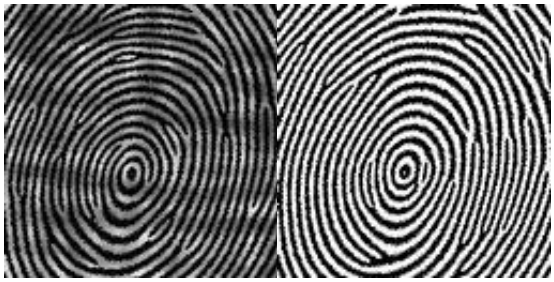

Figure 4: Unfiltered and filtered fingerprints

While this method works well for the U.are.U scanner, it is not an ideal, generalized solution. For example, it would fail to work properly if presented with a data set that had an equal balance of white and mid-grays for valleys. The algorithm would fail to calculate an accurate threshold. A more comprehensive solution is currently a work in progress.

Section 3: Match those minutiae

Minutiae refer to the minute details of a fingerprint, what makes it unique; its code in a sense of the word. There are many ways of classifying minutiae as are listed below in Figure 5.

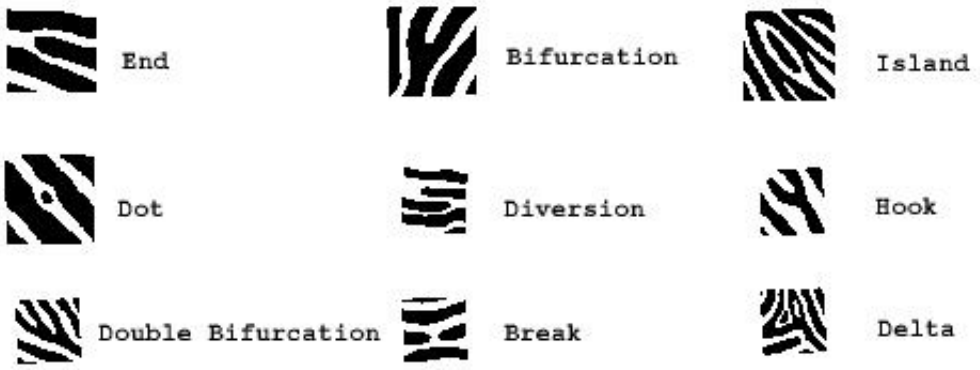

Figure 5: Various minutiae

However, for our matching analysis, we will use only minutiae types ’End’ and ’Bifurcation’. Any of the combinations above can be expressed in terms of End and Bifurcation, and so they represent the 1’s and 0’s of the fingerprint world. When analyzing the image we should only pay attention to black or white areas, properly referred to as ridges and valleys. If we mix the two, a ridge’s End could also be seen as a valley’s Bifurcation. Therefore we must be careful to differentiate between them.

Using the filtered image obtained through methods described in Section 2, we will search for Ends and Bifurcations. Searching for minutiae is done by choosing a starting point (a pixel on the image), and following that ridge (actually a color, in this case black) until an End for Bifurcation is found. When either is found, a mark is made on the image matrix struct depicting what kind of minutiae was found at that x,y position. This continues until the entire image has been processed. The code only searches for minutiae in the ridges, using the valleys for contrast.

After processing the filtered image we are left with a series of x,y coordinates. By comparing these coordinates to those stored locally (or remotely), a search in performed to find a match. The filtering algorithm (of Section 2) and locating minutiae (above) represented two statistically challenging parts of biometric verification. Actually matching fingerprints is the third.

The approach taken in libbioauth is brute force. Using the set of coordinates from the recently scanned fingerprint and those stored locally, a comparison is made by rotating the new fingerprint until match is found or failure is declared. The coordinates are compared 30 pixels in each direction from the images relative center. The coordinate positions of 20 minutiae must match in order for a successful match to be declared. This number, of course, can be raised or lowered to change the exactness required for authentication. However, depending on the quality of the scanner, the conditions it is stored in (extreme hot or cold), and the conditions under which the persons fingerprint is scanned (recently cut or sweaty), the minutiae ascertained after filtering are always slightly different. Regardless of whether the number of recognized minutiae fluctuates, it is important to note their relative location to each other, for the same fingerprint does not change.

Finally, where is all that fingerprint data stored? The tests performed for libbioauth were based on minutiae coordinates being stored in individual files in a arbitrary directory. The version currently under development is based on a database. Rather than searching through files, a single query to the database will be made, and other qualifiers can be placed onto the table schema making it a more versatile solution. The database will also store information such as the users username, the location of their home directory, shell, and so on.

Conclusion

The bioauth(8) solution fits in perfectly with the Linux landscape, demonstrating the versatility a multitude of hackers has brought to the operating system. The ideas and implementations discussed in this paper represent a beginning, not an ending. For example, bioauth only supports one device, the U.are.U. Device support should be improved, and libbioauth should be extended to support other types of authentication, such as retinal scanning or face recognition. The framework has been provided, but the code has yet to be written.

Attention was paid to implementation of biometric authentication under Linux, but not underlying security. Here are a few points not covered extensively in this paper which are of importance.

1. Whether you authenticate via text password or fingerprint, it is extremely important to keep your secret data secure.
2. If the device is externally attached to the computer, or data is stored remotely, communication

should be encrypted between points.
3. The weakest link is between the chair and the keyboard.

Notes

[1] http://www.retina-scan.com/retina_scan_technology.htm

[2] http://www.bioenabletech.com

[3] http://www.digitalpersona.com

[4] http://www.biometrika.it/eng/index.html

[5] http://www.google.com

[6] http://www.neurotechnologija.com/verifinger.html

[7] http://www.linuxdevices.com/articles/AT7145548309.html
This device is based on Linux!

[8] http://www.neurotechnologija.com

[9] This project has its roots in 1996 and work was done on a parallel port scanner manufactured by Siemens.

[10] http://www.wingmanteam.com/usbsnoopy/

[11] http://www-micro.deis.unibo.it/~tartagni//Finger/FingerSensor.html

[12] The images of Figures 3 and 4 do not represent real fingerprints which have not been published due to privacy issues. The images in this paper have been artificially produced, but are similar to the output from the U.are.U.

Acknowledgments

Thanks to Charlie Peck of Earlham College for being a great adviser and providing support and advice during the development process now and many years ago.

I am eternally grateful to my employer, VA Linux Systems Japan, K.K. who sponsored my presentation and continues to support me and the community in all things Linux.

Special thanks to Simon Horman, Jeanne Reeder, and John Reeder for editing assistance.

Trademarks