Hacking fingerprint scanners – why Microsoft's Fingerprint Reader is not a security feature

Mikko Kiviharju

mikko.kiviharju@pvtt.mil.fi

1. INTRODUCTION

Microsoft Fingerprint Reader [13] is a biometrics device targeted for consumer market with a low price (about 40€ in 11/2005). It offers security-like features, such as logging in to Windows and certain type of websites. However, in the security disclaimer [11] it is clearly stated that the fingerprint scanner should not be used as a security feature. As there seems to be very little technical details publicly available *at all* of the product, let alone any tests or anything other than rumors, it remains unclear as to *why* the scanner should not be considered secure. These questions form the main part of this testing work and this paper.

As it turns out, Microsoft has merely repackaged third-party software and hardware (namely that of DigitalPersona), and thus the investigation turns to their products as well. In addition, it is possible to develop third-party software on top of MSFR, but surprisingly not with DigitalPersona's SDKs. An independent Brazilian academics-based company called Griaule has developed an SDK, which can use the MSFR. This SDK provides important insight to the inner workings of MSFR and DP's drivers.

Most existing biometric authentication systems can be described with the following architecture (standard ANSI X9.84) [1]:

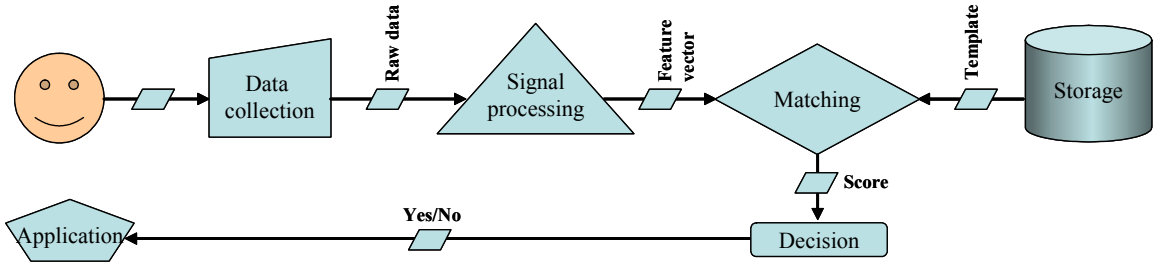

Fig. 1: Biometric authentication system architecture

Of this architecture, the path from signal processing to decision making is usually physically well protected, and the storage well encrypted. This makes the data collection and raw data more tempting targets. The tests performed here concentrate on thwarting each of the two components: data collection (forging a fingerprint to the scanner) and raw data (intercepting the fingerprint scan).

We first introduce the whole picture, technical details and functionality, along with the (non-) usage of cryptography. We then turn to the tests planned to thwart the anti-forging mechanisms, and finally to the tests performed to sniff the raw data travelling to the signal processing.

2. MSFR FEATURES AND FUNCTIONALITY

The MSFR physical device consists of an 512dpi DigitalPersona U.are.U Fingerprint Module [4] scanner, and MS packing. The dpi-rating is an average over the scanning area, since the part closer to the USB-cord is scanned with more accuracy then the "lower" part. This is probably due to anti-forgery reasons. The dpi-rating is taken from DigitalPersona and Griaule documents; tests showed horizontal resolution of over 650dpi, and 350-450dpi in vertical direction. The scanner produces a 8-bit grayscale raw images with an unknown header-type of 64 bytes. Actual scans are 384x289 pixels and "checksums" 384x90 pixels. The scan and the checksum are then transmitted unencrypted over USB to DigitalPersona USB drivers and DP Password Manager software. The drivers are able to rotate (the scanner itself does not identify or rotate) the template, which makes the finger physical direction irrelevant. The image is not 1:1 picture of the fingerprint, but rather an conformal transformation of it. Some other data (likely for anti-forgery purposes) is sent as well.

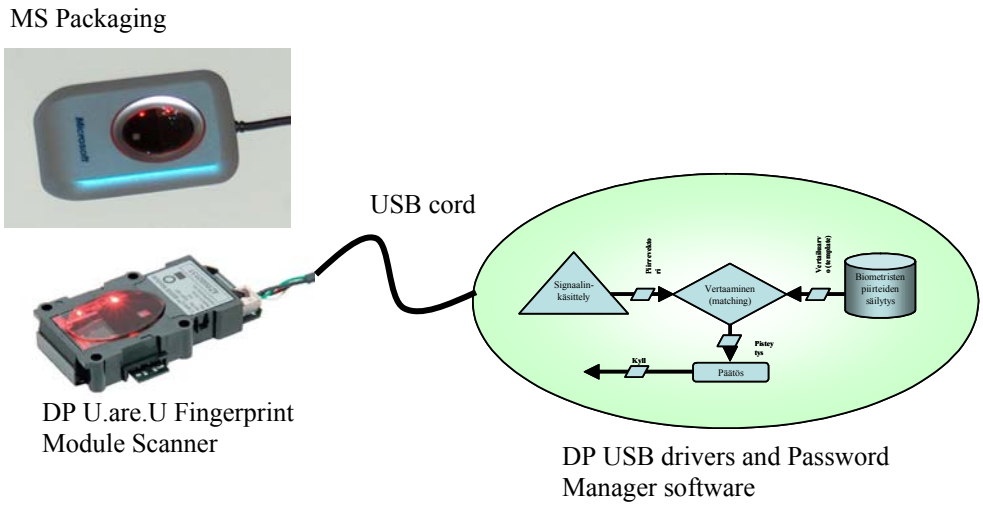

Fig. 2: MSFR components

The MSFR scanner is mainly of optical nature, but it also reacts to certain pressure distributions on the image capture screen, although not heat. The significant pressure distributions are of coarse nature only.

The scanner does not contain much intelligence: it only reports its presence when plugged in, two events on finger placing and finger removing, and the actual scan. The drivers don't input anything relevant (other than USB control data and necessary protocol requests) to the scanner. From the scanner's point of view it is irrelevant, whether the fingerprint is accepted or not. The scanner does, however, make some rudimentary quality checks: if presented with something non-reflective, it will not scan completely (no checksum is sent), but it will scan again (several times occasionally), and then report the fully augmented scan to the drivers.

The following message sequence chart depicts the messages and functionality of the MSFR during different phases of operation: initialization, medium or good quality fingerprint scan, and a a non-fingerprint scan.

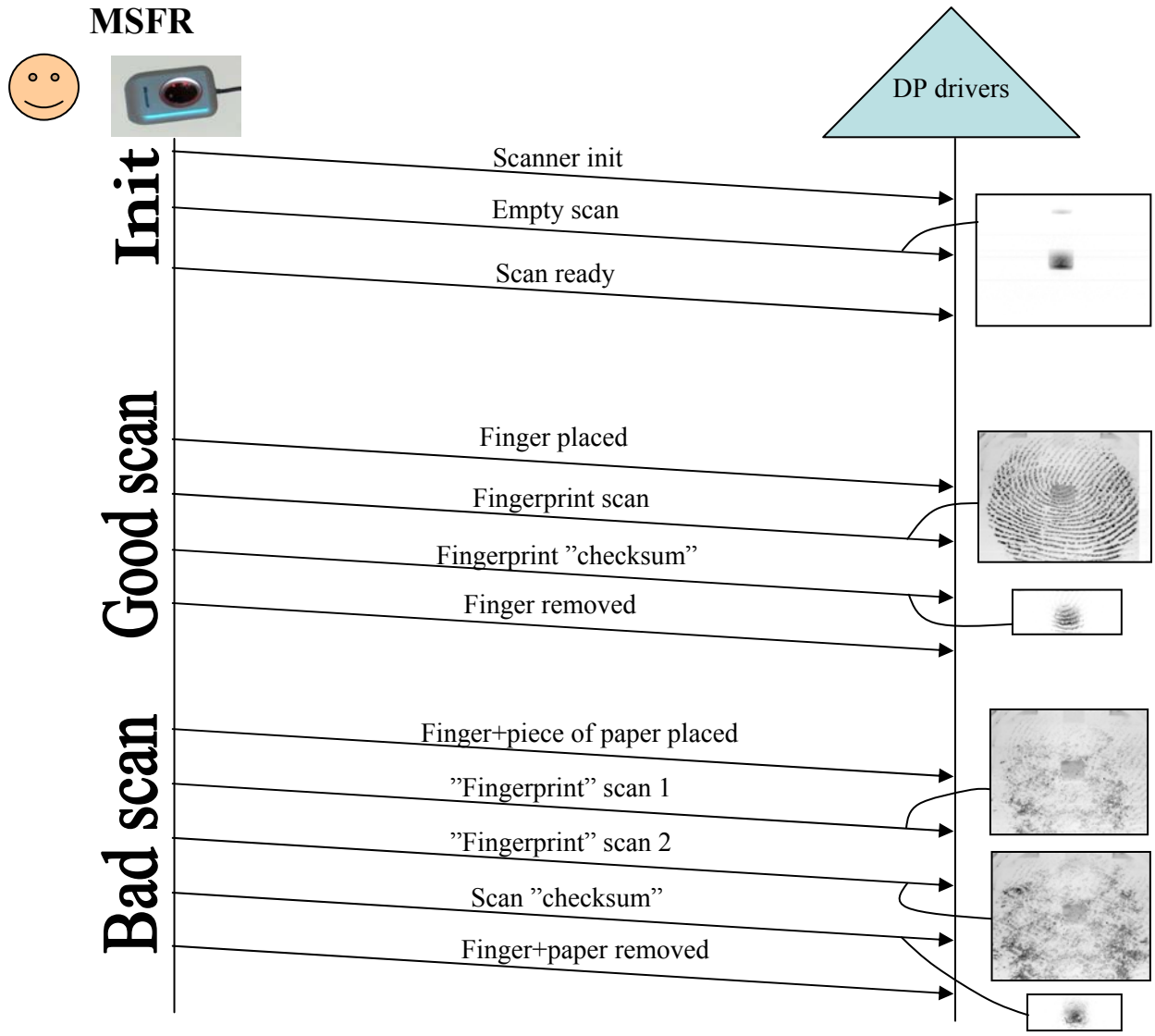

Fig. 3: Messages between MSFR and driver in different scenarios

The different scanning resolutions produce a distorted (or transformed) image. The exact formula for this transformation is not known, but it can be estimated from the Griaule SDK's [8] images and test scans with a grid. The Griaule's screenshot is in figure 4. (Griaule's GrFinger SDK contains sample applications, which show that the SDK automatically corrects the distortion resulting from MSFR, and shows the fingerprint scan as is).

Fig. 4: Screenshot from GrFinger sample software after scanning a fingerprint

The transformation formula estimated from calculating the widths and heights of different scans is

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1{,}153x\left(1 - 6{,}410 \cdot 10^{-4} y\right)^{-1} \\ 0{,}845y \end{pmatrix}$$

Where vector *(x', y')* represents the distorted image coordinates.

As can be seen from the message sequence chart in fig. 3, there is no feedback outside the USB-protocol to the MSFR. (There are plenty of data going towards MSFR, but these turn out to be request messages required by the USB protocol full of used memory areas.) Hence, there can be no cryptographic key-exchange protocol active either. This precludes the possibility that an existing cryptographic protocol would just be badly used. There is *no* crypto used in conjunction with this part of the communication channels.

There are reportedly [7, 16] and easily seen to be DigitalPersona's U.are.U technology inside the MSFR and its drivers. It is also known that with DP's fingerprint scanners the U.are.U is using there are quite decent cryptographic protocols (involving challenge/response and session-keys) and algorithms [3]. The tests also found out that the DP's drivers still possess the cryptographic abilities – they are just not used. The tempting conclusion is that in making a low-priced product MS and DP have settled (for commercial reasons) to less secure functionality to obtain this low price.

3. TESTING THE MSFR FORGING RESILIENCY

Ever since the appearance of silicone fingers molded with gelatine, making fake fingers has become affordable to the layman. It is even longer that the fingerprint reader manufacturers have struggled to recognize all but the most genuine forged fingers. Some early capacitive readers were susceptible to latent fingerprints on the scanner plate, if applied moisture on, for example.

Today, several methods are applied to prevent finger faking, including detecting pressure, temperature and liveliness of the finger as well as the optical properties of the skin.

MSFR is an optical scanner, so it seems possible that presenting the scanner with some sort of image or scan of the actual finger would work. Starting from this, several variations and combinations were tried, none to achieve full recognition. A partial good quality fingerprint was extracted, though, using a 2D-image, but this never reached the matching module, as the checksum was not acceptable. However, the signal processing phase found the scan of "medium quality" and extracted even some features.

The test material included several types of scanned or manufactured images, which were printed to fingerprint size. Most physical testing objects are shown in the figure 5 below.

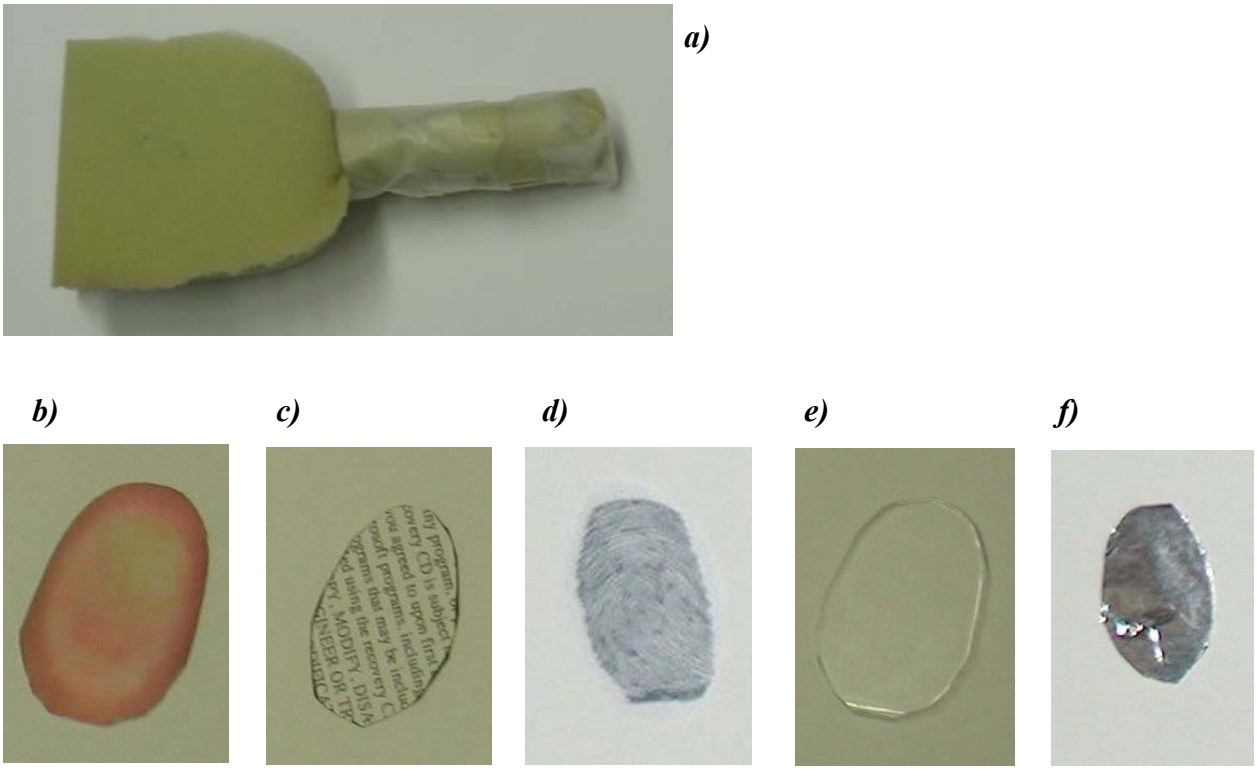

Fig. 5: Physical testing objects

The first test was to access the signal processor, i.e. activate the scanner. There were three actual tests:

- Purely optical: Since this is an optical scanner, there could be a sensor sensitive to intensity of incoming light. This was tested by totally covering the scanning oval, and by covering only those parts that are covered when a finger is presented. None of these produced any scan.
- Pressure-based 1: The next thing attempted was to press with a blunt instrument (the blunt side of a pen), with and without an optical cover (paper) in between. These didn't produce a scan.

- Pressure-based 2: We constructed a "finger" from styrofoam (fig. 5a) by taping the other end to the size of a finger. When pressed with this fake finger (which did *not* contain any excess heat), the scanner reacted by producing a smudgy figure – not anything like a fingerprint, but at least it entered the signal processing part.
- Pressure & optical: wrapped a normal finger with tinfoil. The scanner sensors did not pick up any light (because the tinfoil reflected the light rays into directions past the sensors)

Concluding the scanning activation tests it can be argued that the image acquisition is not purely optical, but doesn't require any heat either. Not just any pressure will activate the scanner, but it has to have a certain distribution. It seems that this distribution is rather coarse, and does not have to resemble an actual finger – this was tested with wrapping thin transparent plastic around a fingertip: this smooths out smaller height differences in the ridges, but doesn't affect optical properties.

The second test involved miscellaneous types of unconventional scans mainly to see what and approximately how is scanned, and also to test if there are any obvious glitches in the driver software. Five trials were performed:

- blank paper
- reflective surface (fig. 5f)
- B/W paper with text (fig. 5c)
- An actual scanned (with a 1200 dpi normal scanner) fingerprint (fig. 5b)
- A high-contrast ridge-lines-only image (BWR) (5d, hand-drawn on top of image in 5b)

None of these produced hardly anything but smudge. A curious thing was noted, though: the scanned text picture *did* contain the text (fig 6, negated and contrast-enhanced) but it seems as though the is not perpendicular to the plate (this agrees with the rest of the evidence: tinfoil reflecting the light somewhere else and the distorted image).

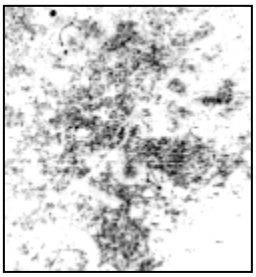

Fig. 6: scanned text image

These focusing or beam-directional issues seem to be related to factoring in the optical and 3D-properties of the skin and the ridges. Making the assumption that altering the focus artificially could enhance the scan resulted in the third set of tests, where the attempt was to get a good quality scan of the ridge-lines image (fig. 5b). The tests involved a piece of transparent plastic film (fig. 5e) to allow the beam to travel through, altering the path of the beam. The combinations (counting from down closest to the scanning plate upwards) used were:

- Film, image (film reflects too much, considering the possibility that the beams are not perpendicular, and only smugde is output)
- Image, film (not enough reflcetion, smudgy)
- Image, folio (too much reflection? The scan was still smudgy)
- Image, film, folio

- Film, silly putty
- Film, small patch of tin foil, silly putty

The last two combinations proved to be fruitful: silly putty provided the ideal optical background for any sorts of prints to be clearly scanned. However, this did not remove the checksum guard yet. The checksum was attempted to be thwarted by a small piece of tinfoil placed near the "focal point", where the checksum is extracted from. This produced a fairly plausible looking checksum, but obviously not good enough for the DP's drivers.

After a few trials of the last combination, a partial fingerprint appeared on the scan (fig. 7). GrFinger SDK even found 10 to 20 minutiae points, so it can be argued that this combination penetrated the system inside signal processing, possibly even through it.

Fig. 7: Fake finger scan and its checksum

Even though the third set of tests was unsuccesful in penetrating through the system, it did reveal yet another anti-forgery mechanism. The MSFR device sends after each scan it considers succesful, a smaller and somehow masked negative image of the middle portions of the finger. We named this image the "checksum", and it seems to be taken from the area marked by a small (15% of the image width) rectangle in the middle of the main image. This checksum seems to be scanned from a different angle or similar, since the reflective properties are different in the checksum from those in the actual scan.

This checksum feature is not visible through GrFinger's user interface, nor the exposed functions. It can be seen only by snooping the USB driver / line, and would be hidden, if the line was encrypted. In a way, Microsoft decisions have affected the security of the third party's hardware as well.

4. GRIAULE'S ENCRYPTION

Even though not directly related to MSFR, it is instructive to look at the GrFinger SDK encryption as an example how certain cryptographic assumptions don't apply to biometric information.

Griaule offers three versions of its SDK, one free and two commercial ones. Only the most expensive one allows one to save images unencrypted, the light and free versions encrypt the image. However, the software does allow one to read saved and encrypted images back again to the software. As it turns out, key management has not been thought thoroughly over: only one key (the same throughout scans and even reboots) is ever used to encrypt the images.

If this one key is well hidden (encrypted itself, or such), this doesn't pose a problem for normal images. However, biometric information has the property, that no two readings are ever exactly alike, nor are they very far apart from each other. This means that if the encryption works as a stream cipher (or a block cipher in OFB- or CTR-mode) the background pixels of two subsequent scans will be encrypted the same, but the actual ridge lines will have slightly different color values, and they will be encrypted differently. Also, the first difference only affects that pixel and not the rest (a property of stream ciphers). Thus it is sufficient to look at the difference between two subsequent scans, as in the figure 8.

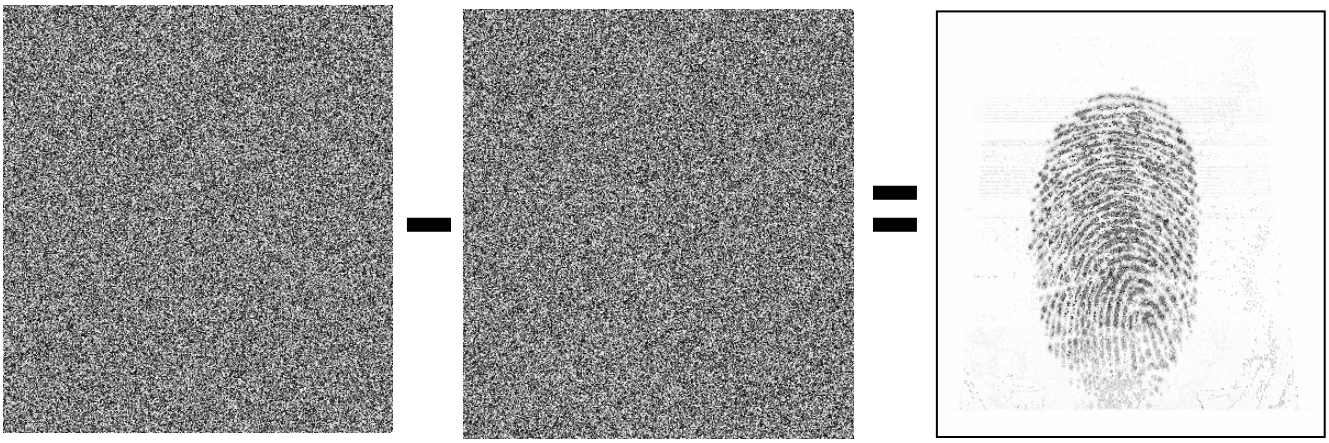

Fig. 8: fingerprint extracted from two different encrypted scans of the same finger

This type of example shows that in addition to being able to use GrFinger free version images just as well as the most expensive version, one should be extra careful and mindful of the particularities of biometric data.

5. SNIFFING THE MSFR USB CHANNEL

To be able to actually test what the MSFR is actually sending, there are two possibilities: to reverse-engineer the DigitalPersona software and to intercept the USB traffic going from the scanner to the drivers. The first option has the positive sides of providing also knowledge of the web password protection functionality, but it is highly time-consuming and legally problematic. The latter option works better, but there is very much filtering and conversions to be made and some (small) uncertainty as to the actual data in the USB line.

In an ideal situation one would place a hardware USB-sniffer between the MSFR and the computer. However, usable hardware USB sniffers are expensive or difficult to come by. Two alternatives were considered:

- A normal hardware PS/2 keylogger (such as in [9]), appended in both ends with PS/2-USB converter. This didn't work, though, the wiring is not consistent with the USB wiring needed for MSFR, which did not even start up.
- A USB-keylogger (i.e. in [10]). This could work in principle, but current models only have 128 KB of memory. The MSFR initialization phase alone produces at least 130 KB of data, filling up the keylogger with irrelevant data. The keylogger can be used in overwrite mode, but the fingerprint scan (109KB) is followed immediately by the checksum (34KB), so there would be – if one could snatch the keylogger right after the scan – at most 86% of the actual fingerprint available. In addition, the USB protocol liveliness messages will overwrite even this in longer timescale (days). Also in the tested USB keyloggers the hardware wiring was incompatible.

Instead of a hardware sniffer we resorted to a software sniffer, "USB Sniffer" available at [15]. The sniffer adds an extra driver and traps calls between the USB host client and USB driver in the USB System SW (the USBD interface as described in [2] ch. 5.3, see figure 9).

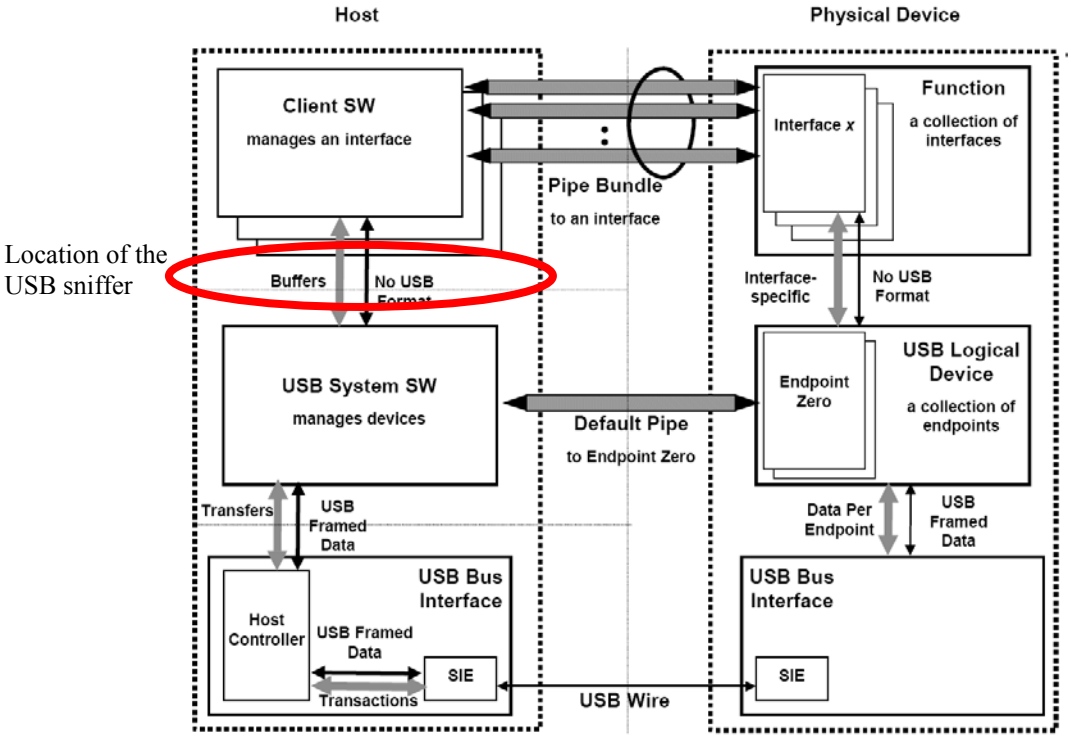

Fig. 9: USB architecture

The USB sniffer expands the USB protocol relevant codes, but the payload in represented in ASCII-formatted memory dump. To translate this, it was necessary to add a bit of conversion routines to the sniffer to convert this dump to binary bitmap files. Also, MS Excel was used to check small portions, whether they formed ASCII or UNICODE text.

The USB protocol produces a lot of data in the USBD interface (the total amount of logs generated during the tests was in excess on 20MB), most of which has no relevance to the fingerprint recognition process. For example:

- according to the USB protocol, the device may not sent unsolicited messages, but everything must be first requested. This is circumvented by having the driver send a request every time a message (of unsolicited nature) is received, keeping the device "alert".
- For some reason, the request must be at least as large as the expected response – at least in the USBD interface. For image scan requests older scans are used (sent back), or if not available, memory dumps encoded in ASCII.
- A number of control messages are sent surrounding all activity, these are in return requested separately.

It helps somewhat that the USB payload is transferred in different channels (what USB calls "pipes") from control data. However, when establishing the lack of any cryptographic key exchange, it cannot be assumed that the DP's driver would not use control channels for this purpose, so each channel has to be checked.

6. LACK OF CRYPTOGRAPHIC PROTECTION OF THE FINGERPRINT IMAGE

It was not anticipated at first to find unencrypted images within the USB-stream, after all in [7] it was mentioned that MSFR uses the XTEA-algorithm [14] for protecting the data in the wire. Rather some examples of encryption were sought in order to determine, whether the same symptoms of lack of session keys of bad encryption mode that were discovered with Griaule, would apply with MSFR as well.

The actual test was simple: the sniffer was turned on, the MSFR scanner plugged in, and some trials with registered and unregistered fingers were made. The trials were spaced apart several seconds, so they could easily be identified from the USB sniffer log.

The log file format identifies the direction of the messages and its type. From these, it was easy to identify the 65KB blocks travelling towards the DP drivers. It became apparent later on that the scanner sends two pictures (the first in two parts): the main scan and the checksum.

The blocks could easily be seen to be unencrypted: any decent cipher produces bytes that vary evenly between all hexadecimal values from 0x00 to 0xFF. The suspected image blocks contained many continuous blocks of 0x00, and almost none of the higher values. In addition, certain patterns seemed to be repeating every 384 bytes. The natural guess here, of course, is that the scanner is sending 8-bit grayscale images in raw (uncompressed) format, as scanners habitually do.

We next combined the several parts, and transformed them into their original bytes (using a tailored C++-program), and opened them with PaintshopPro. When presented with a RAW-type image, PSP asks for the dimensions of the picture, and the guess was 384 pixels wide and large enough height (say 1000 pixels). A resulting greyscale image came out, which contained a clear, but distorted image of a fingerprint, followed by what appeared to be a negative portion of the scan. The image had a black background and it seemed to be wrapping around – there was an edge frame right in the middle of the image. This turned out to be due to an image header of 64 bytes. When this header was removed with HexEdit [6], the image as well as the checksum were poised in the middle. To aid readability the image was turned into its negative, ridding us of the black background. The resulting image is shown in figure 10.

Fig.10: Sniffed unencrypted fingerprint scan and its checksum

Now it was clear that the image is passed unencrypted over the USBD interface, but it is still unknown whether it is scrambled in the actual USB cable. It is possible that the DPs drivers decrypt the image and pass it unprotected upwards to the feature extraction. However, should this be the case, the problem goes far deeper than just Microsoft's product. Also, there is the case of missing cryptographic protocol: one would think that key management would have to involve components above the USB system software level. Key management was something that was not found, however.

7. LACK OF CRYPTOGRAPHIC PROTOCOLS IN THE USB CHANNEL EXCHANGES

The search for a cryptographic key exchange was based on finding something uniformly variable between scans or even sessions, which would be passed both ways and which would also have sufficient length for a cryptographic key (128 bits = 16 bytes). Basically every message was suspect, but it soon became obvious that the USB control messages mostly didn't have any room for that lengthy payloads. There was an exception of a nine-part 1904-byte message sent in the initialization phase, but frequency analysis showed that it was not encrypted nor random (which a proper key should be) – and it repeated identically during other initialization sequences. This left three other places to search:

- 64-byte payload messages meant obviously for event-passing by the scanner
- 64KB output USB-requests
- Image headers

Each of the 64-byte MSFR event messages were classified according to their contents, and even though the DP driver sends varied messages in the initialization phase (only one, though), the

MSFR never sends back any more than three types of messages. The contents of these messages do not vary over scans, session or reboots, which they should, if they were used for key establishment.

The 64KB output requests contain ASCII-formatted memory dumps (of zeroes) or images from used buffers two scans back. There are no irregularities in there that could be attributed to anything random or encrypted.

The image headers could be categorized into three groups:

- the initial empty scan headers
- the actual scan headers
- the checksum image headers

Within these categories the headers differ very little. Initial scan headers are all alike. The other headers incorporate a sequence number, which is incremented every time a new image is formed. In checksum headers this is the only difference. Scan headers have additional differing byte that seems to be varying within four units. This means that not even in the image headers is there any room for carrying anything session- or scan-dependent cryptographic information.

Since there is very little that identifies the sender of the scan or even verifies the scan as legal, it is also easy to replace the MSFR with a custom-made device, and replay the captured images. With simple sequence numbering and recordable initialization sequence, there is no way the USB drivers can verify the sender. This means that even though the fingerprint can be hard to reproduce to the scanner, the scanner *image* is very easy to reproduce to the driver.

The aforementioned properties lead to a following replay-attack where there is no need to actually record anything from the target computer: one can produce a template with one's own computer, and with i.e. a latent fingerprint image compute the scans and checksums needed by the replay and then combine the template with the actual data.

A very simple setup (a laptop and a male-to-male USB-connector cable) is required for the actual playback. The attacker laptop needs to have a special software made for reading USB-logfiles and playing them back, but this type of software is not hard to modify from existing USB sniffers.

8. CONCLUSIONS

We have shown here that Microsoft Fingerprint Reader lacks both cryptographic protocols and encryption of the scanned image in at least one crucial point on the path from the finger to feature extraction. This leads to several things: first and foremost the victims fingerprints are easily captured and stolen. Secondly replay attacks agains this configuration do not require a fake finger *nor* logfiles from the target, but a custom-made USB-device will do. Thirdly, MSFR unencryption reveals some anti-forgery strategies used by DigitalPersona elsewhere as well.

We have additionally tested and suggested some properties of the scanning device. Although it we couldn't produce a forgery that would have been accepted as a real finger, we were able to push through a partial scan, and we feel that as the checksum-feature is known, the information to build a decent fake finger is very close indeed.

Even though MSFR can not be categorized as a security device, it is used as user credentials for logon. There is a small positive side here, namely the domain administrator is unlikely to log in with MSFR. The DigitalPersona Password Manager uses the fast switching functionality, which is disabled in domain members [5], and the software cannot log in anybody to a domain. Thus

any breaches to user accounts in this way are in principle contained (but of course a skilled hacker can find other means to promote his status from a local used to domain admin more easily than from unauthorized user).

As the Griaule example shows, biometrics is a field with its own special characteristics that should be taken into account when building a secure biometric system. Security by obscurity proves yet again to be an insufficient measure – it can be argued that obscurity enhances security, but it does not create security.

9. REFERENCES:

[1] American National Standards Institute. **ANSI X9.84 – 2001.** In Biometric Information Management and Security. 2001.

[2] Compaq, HP, Intel, Lucent, Microsoft, NEC, Philips. **Universal Serial Bus Specification, Rev. 2.** In http://www.usb.org/developers/docs/usb_20_02212005.zip. 27.4.2000.

[3] DigitalPersona. **DigitalPersona SDK.** In http://www.digitalpersona.com/resources/downloads/SDK8-05.pdf . 2005.

[4] DigitalPersona. **U.are.U Fingerprint Module.** In http://www.digitalpersona.com/products/uauFingerprintMod.php. 2005.

[5] "Doug". **Fingerprint reader question and remark.** In http://groups.google.fi/group/microsoft.public.windowsxp.hardware/browse_thread/thread/56d8fdb314b0075c/17e5aa75c5f6ea1f?q=%22digital+persona%22+domain&rnum=1#17e5aa75c5f6ea1f. 26.5.2005.

[6] Expert Commercial Software. **HexEdit 3.0.** In http://www.expertcomsoft.com/download.htm. 1.1.2005.

[7] GlobeTechnology. **GlobeTechnology: Microsoft Fingerprint Reader.** In http://www.globetechnology.com/servlet/story/RTGAM.20041115.gtfingernov15/BNStory/TechReviews/. 24.11.2004.

[8] Griaule fingerprint recognition. **Griaule Fingerprint Recognition SDK.** In http://www.griaule.com/en/fingerprint_recognition_sdk.php. 2005.

[9] KeyGhost. **Keyghost (Software Free) Keylogger for Home Use.** In http://www.keyghost.com/keylogger.htm. 12.1.2005.

[10] KeyGhost. **New KeyGhost USB Keylogger 128KB.** In http://www.keyghost.com/USB-Keylogger.htm. 29.9.2005.

[11] Microsoft. **Getting Started. Microsoft Fingerprint Reader.** In http://download.microsoft.com/download/1/3/9/139a8c30-34cc-4453-a449-7a1c586a3ae5/Fingerprint_Reader.pdf. 20.4.2004

[12] Microsoft. **Microsoft Fingerprint Reader Technical data sheet.** In

http://download.microsoft.com/download/9/a/8/9a8092c2-9da7-4ab8-a503-1be94633d112/TDS_FingerPrintReader_0410A.pdf. 2004.

[13] Microsoft. **Microsoft Mouse and Keyboard Hardware – Fingerprint Reader.** In http://www.microsoft.com/hardware/mouseandkeyboard/productdetails.aspx?pid=036. 2005.

[14] Needham, R; Wheeler, D. **TEA extensions.** In Technical report, Computer Laboratory, University of Cambridge. October 1997.

[15] Papillault, B. **USB Sniffer for Windows 98, 98SE, 2000 and Windows XP.** In http://benoit.papillault.free.fr/usbsnoop/doc.php.en. 21.5.2003.

[16] "swg". **Biometric Fingerprint Reader.** In http://groups.google.fi/group/microsoft.public.development.device.drivers/browse_thread/thread/2115477b3cc7e4f/4d8978665b0d9e12?q=DigitalPersona+SDKs&rnum=1#4d8978665b0d9e12. 18.4.2005.