Chapter 5

User authentication

Read Chapter 3 (User Authentication) of the text book.

This topic is part of the domain of *System security*.

5.1 Introduction

Authentication is used to

- control access to the system,
- enable a personalized environment,
- determine the privileges of the user,
- track the user (log files).

Three typical steps involved when accessing a secure system (e.g. a typical login on a computer):

1. Identification
   Tell the system who you are (e.g. user name).
2. Authentication (or verification)
   Prove to the system that you are who you claim to be (e.g. password).
3. Authorization
   The system determines which access you have to which resources.

The third step is treated in the chapter on *access control*.
Note: not all these steps are always necessary (e.g. access control requires you to be genuinely authorized – this does not always require proper identification).

Question: give an example where access can be achieved without proving who you are.

- withdrawal from ATM machine,
- access to public transport (train, bus),
- access to concerts / theme parks / ...
- etc.

Authentication factors. User authentication is based on something the user:

- knows (password, PIN),
- possesses (key, token, smartcard),
- is (static biometrics) (fingerprint, retina, face),
- does (dynamic biometrics) (voice, sign, typing rythm).

Definition 7 (Two-factor authentication) *Authentication using two* different *categories.*

Example 5.1.1 (Two factor authentication) *Authentication using smartcard (possesses) plus pincode (knows) is two-factor authentication (two different factors used).*
Authentication using password (knows) plus mother's birthday (knows) is not *two-factor authentication (only one factor used).*

5.2 Password-based authentication

Password are memorized by user and stored by the system. Storing plaintext passwords is considered vulnerable – a system administrator can see the passwords, and so can anyone who successfully hacks the system. To reduce these risks, the hashed images of the passwords are stored instead of the passwords themselves.

5.2.1 Aside: hash functions

A cryptographic hash function is a function that takes a plaintext of variable length, and produces a "fingerprint" of the plaintext (a bitstring of fixed length, e.g. 128 bits).
Cryptographic hash functions have the following properties:

- given m, easy to compute $h(m)$.
- a small change in $m \Longrightarrow$ a big change in $h(m)$.
- (*preimage resistance*) given $h(m)$, intractable to find m.
- (*second preimage resistance*) given m, intractable to find m' such that $h(m) = h(m')$.
- (*collision resistance*) given $h(m)$, intractable to find m' such that $h(m) = h(m')$.

notation: $h(m)$.

Note: *Cryptographic hash functions $\neq$ computer science hash functions!*
A Computer Science Hash Function is a function that maps elements from a large, possibly infinite domain into a small (finite) range. Due to the Pigeon Hole principle, this means that there exist different input values which will be mapped to the same output value (a collision). It is considered desirable for CS hash functions to map the inputs as evenly as possible over the range. There is no equivalent property for cryptographic hash functions.

Using hash functions for passwords:

System keeps file of $(id, h(pw))$ pairs.

1. User provides id and pw.
2. System calculates $h(pw)$ and verifies if $(id, h(pw))$ exists.

5.2.2 Vulnerabilities

- Password can be guessed,
  - brute force attack, dictionary attack, popular password, password derivable from owner data, default system password.
- Password stored at system can be disclosed,
  - plain (or obscured) passwords in password file,
  - hashed passwords in password file $\rightarrow$ offline dictionary attacks,
  - physical access: reboot with e.g. linux live cd.
- Password stored by user can be disclosed,
  - password written on note,
  - password shared with colleagues,
  - similar/same password on multiple systems.
    Question: who reuses their (Facebook) password elsewhere?
- Password in transit can be intercepted.
  - protocol vulnerability,
  - replay attack of encoded password.
  - shoulder surfing
- Multiple users with the same password are obvious in the password file (same hash value).

Note that contemporary approaches to cracking passwords hardly ever rely on bruteforce, but use rainbow tables, wordlists and rules to generate guesses.

Example 5.2.1 (Bruteforce vs wordlists) *Consider the password "`Facebook2013Hugo`". It has 16 characters from `[A-Za-z0-9]`, thus 62 possibilities per character. That gives* 16^62 *possibilities to consider for a brute force attempt. On a computer that can process 30,000,000 guesses per second, this would take about* $479 * 10^{54}$ *years.*

A word list that contains well-known internet services (google, gmail, facebook, twitter, linkedIn, ...), user names (including Hugo), and some rules for capitalisation and adding numbers will find the right combination within seconds on the same computer.

Using well-known substitutions (e.g., "`F4c#b00k2oi3Hug)` ") doesn't really help against this type of attack – the attacker simply adds these rules to the rule list, and again easily finds the correct password.

5.2.3 Possible countermeasures

- requirements on password (size, character set, freshness),
  Problem of rule enforcement: if we want to prevent users from selecting a "bad" password (from some dictionary of bad passwords D), we have the problems of space (for storing the dictionary) and time (for verifying admissible passwords). A proposed solution is to use a *Bloom filter*, see below.
- password generation (automatic or mentally),
  - mentally (e.g. using l33t-substitution in your name).
    problem: easy to bruteforce if trick is known (recall Kerckhoffs principle: no security through obscurity).

Compiled from various sources and editted by Sjouke Mauw and Hugo Jonker. For UL internal use only. ©2007-2013, S. Mauw & H. Jonker.

    - automatically generated passwords:
      problem: hard to remember, solution: generate pronouncable passwords.
      (recall security dilemma 2: security vs. usability)

- password salting, (see below)
- user training,
- use slow hash function or use *key stretching* (see below),
- limit number of unsuccessful authentication attempts.
  In this case, a recovery strategy is needed – cf. Security Dilemma 2 (security vs usability).

Key stretching. Key stretching is ensuring that computing a hash takes a considerable amount of time, for example by applying a hash function very often. This time should be tolerable for a user logging in, e.g. 1 sec.

Key stretching affects an attacker far harder than a genuine user: an attacker (with the same hardware) will need 1 sec to try out 1 password, instead of being able to try thousands of passwords per second. Thus this defends against automated password-guessing attacks.

Password salting. Procedure for authentication with salted passwords:

1. System keeps file of $(id, salt, h(pw, salt))$ tuples.
2. User provides id and pw.
3. System looks up in the file which salt corresponds to user id, then calculates $h(pw, salt)$ and verifies if this is equal to the hash in the password file.

Goals of password salting:

- increase time needed for offline dictionary attacks,
- duplicate passwords in file not visible,
- duplicate passwords of a user on different systems not visible.

Note: generate *salt* randomly – do *not* take characters from *pw* to serve as *salt*! (observed in real life).

Security Dilemma 7 *Security vs. obscurity.*
"security by obscurity" = the use of secrecy (of design, implementation, etc.) to provide security. E.g. hiding passwords in a file system; secret URL-extensions.

Kerckhoffs' principle (1883): "security of a system should depend on its key, not on its design remaining obscure".

Preventing bad passwords with Bloom Filters

Bloom filter: tests whether an item x is in a set D. Answer: yes/no.
Bloom filter test for bad passwords: test whether the new password x is in the set of forbidden words D.

- Given:
    - T: a boolean array of length N, initially all false
    - D: a set of bad passwords,
    - k hash functions* $h_1, \ldots, h_k \colon D \to \{0, \ldots, N-1\}$
      * Computer Science hash functions, not cryptographic hash functions (see Chapter 2).

- Initialization: calculate for every $d \in D$ the hashes $h_1(d) \dots h_k(d)$. For each $d \in D$, set $T(h_i(d))$ to true (for all $1 \leq i \leq k$).
- Password test: reject a proposed password x if $T(h_1(x)) \wedge \dots \wedge T(h_k(x))$. Clearly all words from D will be rejected.
- Note:
  - some words that do not belong to D will also be rejected (*false positives*).
  - No false negatives (by construction).

Example 5.2.2 (Bloom filter toy example) *Suppose we want to test whether something is an animal.*

- *Initially,* $Animals_1 = \{ape, bear, cat\}$.
- *We pick* $N = 10$, *so* T_1 *is a boolean array* $[0..9]$.
- *We pick one hash function* $h_1(x) = |x|$ *(length of the word x).*

Note that $h_1(ape) = h_1(cat) = 3$ *and that* $h_1(bear) = 4$.

Therefore, $T_1 =$

0	1	2	3	4	5	6	7	8	9
			true	*true*					

.

Testing the $Animals_1$ *Bloom filter with the words* `cat, dog, tree, spider`*:*

word	*h1(word)*	*T1[h1(word)]* (*i.e., filter claims word* $\in D$*?*)	*correct?*
ape	*3*	*true*	*yes.*
cat	*3*	*true*	*yes.*
dog	*3*	*true*	*no.*
tree	*4*	*true*	*no, false positive.*
spider	*6*	*false*	*no, false negative.*

The filter incorrectly thinks a tree is an animal. Moreover, the set $Animals_1$ *does not include the animal "dog", nor the animal "spider". So let's build a better filter, and add "dog" and "spider" to the set of animals:*

- $Animals_2 = \{ape, bear, cat, dog, spider\}$.
- *We keep* $N = 10$, *so* T_2 *is a boolean array* $[0..9]$ *like* T_1.
- *We keep* h_1.
- *We use a second hash function* $h_2(x) =$ *"# limbs of x".*

We now have the following:

word	*h1(word)*	*h2(word)*
ape	*3*	*4*
bear	*4*	*4*
cat	*3*	*4*
dog	*3*	*4*
spider	*6*	*8*

So, $T_2 =$

0	1	2	3	4	5	6	7	8	9
			true	*true*		*true*		*true*	

.

Recall: Bloom filter says $x \in Animals_2$ *if* all *hash functions* $h_i(x)$ *are true in* T_2. *Thus, the filter based on* T_2 *is tested as follows:*

word	*h1(word)*	*h2(word)*	*T2[h1(word)]* $\wedge$ *T2[h2(word)]* (*i.e., filter claims word* $\in D$*?*)	*correct?*
ape	*3*	*4*	*true*	*yes.*
bear	*4*	*4*	*true*	*yes.*
cat	*3*	*4*	*true*	*yes.*
dog	*3*	*4*	*true*	*yes.*
spider	*6*	*8*	*true*	*yes.*
tree	***4***	***0***	***false***	***yes.***
car	***3***	***0***	***false***	***yes.***
airplane	***8***	***0***	***false***	***yes.***

Note: this example, like all others treated for this course, is a toy example to help you understand a Bloom filter's operation.
Real-life Bloom filters operate slightly differently:

Example 5.2.3 (Google Chrome) *The Google Chrome browser uses*[1] *a Bloom filter in the "Safe Browsing" tests. It uses 20 hash functions, and its table has a size of between 244 kilobyte and 3 megabyte.*

5.2.4 Concluding passwords: some anecdotes

American Express requires users to create a PIN (usually, a 4-digit password) when activating their card over the phone. A friendly suggestion is to use your mother's month/day. Apparently, the automated response system will *reject* PINs that are not valid month/days. See link[2] for a story what it takes to have a random PIN.

Review questions.

- Which security dilemmas come into play for this restriction on PINs?
- How many numbers are accepted by the automated system?

Some companies (Microsoft Hotmail[3], American Express[4]) only limit the length of passwords – only the first handful of characters of long passwords will be considered. Sometimes, this is even poorly implemented – American Express apparently limits password length on the "set password" page, but not on the "login" page. This led to confusion and multiple password resets.

Review questions.

- Which security dilemma(s) come into play for these restrictions on password lengths?
- The story of the second link (as explained above) highlights a secondary security/usability problem. What is this problem?

Finally, here[5] is one list of common passwords (there are others).

Review question. Is one of your current passwords on there?

[1] `http://src.chromium.org/viewvc/chrome/trunk/src/chrome/browser/safe_browsing/bloom_filter.h?view=markup`
[2] `http://it.slashdot.org/comments.pl?sid=3135655&cid=41417803`
[3] `http://thenextweb.com/microsoft/2012/09/21/this-ridiculous-microsoft-longer-accepts-long-passwords-shortens/`
[4] `http://it.slashdot.org/comments.pl?sid=3135655&cid=41417953`
[5] `http://www.whatsmypass.com/the-top-500-worst-passwords-of-all-time`

Compiled from various sources and editted by Sjouke Mauw and Hugo Jonker. For UL internal use only. ©2007-2013, S. Mauw & H. Jonker.

5.3 Token-based authentication

Consider:

- embossed cards
- memory cards (and dongles)
  - magnetic stripe
  - electronic memory
- smart cards (have processor)
  - contact
  - contactless

  Smart card authentication can be static (similar to memory card), through a dynamic password generator (e.g. every minute $\rightarrow$ synchronization problem), or based on a challenge-response protocol.

5.4 Biometric authentication

There exist various forms of (sufficiently) unique properties of a human:

- facial
- fingerprints
- hand geometry
- retinal pattern
- iris
- signature
- voice

Biometric systems distinguish between:

- enrollment phase
- verification use
- identification use

These three different uses of the system are shown in Figure 5.1. A password-based system will give a yes/no answer, while a biometric system will give a matching score. By determining a matching threshold, the system can be configured as to meet the required security level.

This means that biometric authentication systems can be tweaked from very strict (almost no false positives) to very loose (almost no false negatives). This is a specific form of Security Dilemma 2:

Security Dilemma 8 *In biometric authentication systems there is a trade-off between the probability of false positives and false negatives.*

(See the figure at page 95 of the text book.)

The *cross-over error rate* is defined as the rate where the false positive rate and the false negative rate are equal.

Compiled from various sources and editted by Sjouke Mauw and Hugo Jonker. For UL internal use only. ©2007-2013, S. Mauw & H. Jonker.

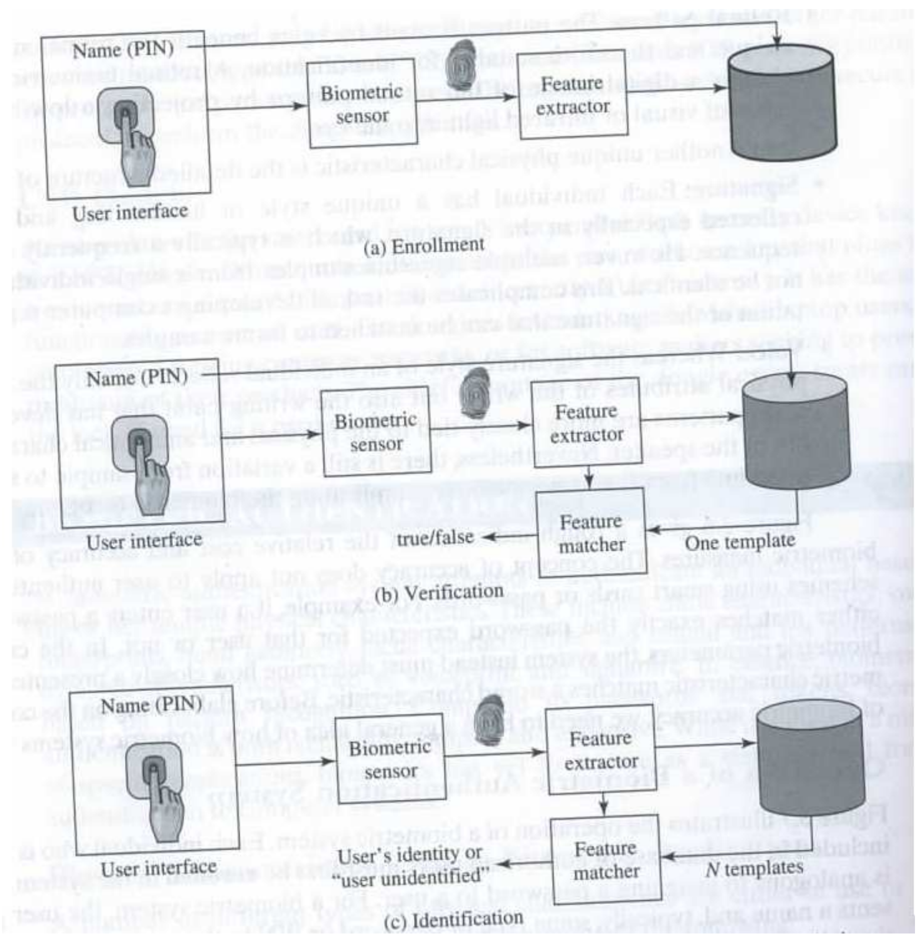

Figure 5.1: Phases of biometric systems (Stallings & Brown, pg. 94)

5.5 Exercises

Exercise 1.
Provide arguments for security through obscurity vs. security through scrutiny.

Exercise 2.
A bank will allow withdrawals from your account when you present a valid passport. On the other hand, a country will not allow you entry with a valid ATM card + pin code.

(a) Which step is the one the bank cares about to access your account? And which step does the country require for entrance to the country?

(b) Why will the bank accept the country's passport, but the country not the bank's PIN+card?

Exercise 3.
Consider a user "Hugo Jonker" who has liked the website BertIsEvil.com for 10 years. Consider furthermore the below password templates:

1. An abbreviation of a sentence. Example: `BiEdChBmFwFaDnAl`
   ("BertIsEvilDotCom Has Been My Favourite Website For A Decade Now At Least")

2. A random string. Example `%z9o&^a‘b+S|q6}L)@`

3. Name + a year in reverse. Example `9002oguH`
4. An abbreviation (as before), using 1337-substitutions. Example `8i3Dc4bmF\/\/4@dN@l` (the "BertIsEvil" password using well-known substitutions)
5. Full name of the user. Example `Hugo Jonker`

(a) List these templates in order of the least number of passwords to most number of passwords possible to generate for the given user.

Instead of using any of the given password templates directly, Hugo uses one of the templates to construct a password, then hashes that password using a well-known hashing function (e.g. using `md5` or `sha1`) and uses the output of the hash as a password.

(b) Against which attackers does this help?

Exercise 4.
Facebook allows 3 forms of your password:

1. How it was typed when last changed,
2. With the first character capitalised,
3. With the case of each character flipped.

(According to Facebook, the second form is to accommodate those phones which capitalise the first character of a word. The third form is to accommodate users who accidentally have CapsLock turned on.)

(a) Which security dilemmas play a role here?

Consider three classes of passwords: simple: 5 characters, `[a-z]` only; average: 8 characters, `[A-Za-z0-9]`; and strong: 12 characters, `[A-Za-z0-9]` and the following 32 characters:
`+~!@#$%^&*()_|{}:"?><‘,./;’][\=-`

(b) For each class, give the number of passwords contained in the class.

(c) For each class, give the number of passwords Facebook distinguishes.

(d) Do you consider Facebook's policy good or bad? Explain your reasoning.

Exercise 5.
During the 2008 USA presidential campaign, the Yahoo e-mail account of Sarah Palin, vice-presidential candidate, was hacked. Knowing her zipcode, birthdate and the answer to the security question (where did she met her spouse - which could be found on the Web) sufficed to reset her e-mail password and access her e-mails.

(a) Is the described password recovery mechanism secure? Why (not)?

(b) Keeping in mind the various encountered security dilemma's, propose a better password recovery mechanism.

Exercise 6.
Describe two situations, in which an identification system would aim to minimize the false negatives instead of the false positives.

Exercise 7.
Explain whether each of the following is two-factor authentication or not:

Compiled from various sources and editted by Sjouke Mauw and Hugo Jonker. For UL internal use only. ©2007-2013, S. Mauw & H. Jonker.

(a) login to internet banking using password and your birthdate.

(b) login to internet banking using a password, after which a pincode is send to your mobile, which you need to enter to gain full access.

(c) a door which requires two keys to open.

(d) a door which requires a key and an iris scan.

Exercise 8.
(Bloom filters) Consider

- a dictionary $D = \{carrot, root, rabbit, hare\}$.
- the hash function $h1\colon \{a, \ldots, z\}^* \to \mathbb{N}$, which takes the numerical value of each letter (a=1, b=2, etc.) and adds all these numbers together modulo 26. Example: $h1(hare) = 8 + 1 + 18 + 5 \bmod 26 = 6$.
- the hash function $h2$, which works the same but skips every second letter (e.g. $h2(hare) = h1(hr)$).

(a) Give the filter table T for a Bloom filter based on D and these two hash functions.

(b) Consider the string $m =$ `kirchberg`. Show whether this string is accepted or not.

(c) Describe a hash function $h3$ such that a new Bloom filter based on $D, h1, h2, h3$ will not reject m. Show that m is not rejected in this new Bloom filter.

Exercise 9.
(Bloom filters) Consider

- a dictionary $D = \{programming, helps, with, homework\}$.
- the hash function $h1\colon \{a, \ldots, z\}^* \to \mathbb{N}$, which takes the numerical value of each letter (a=1, b=2, etc.) and adds all these numbers together modulo 26. Example: $h1(hare) = 8 + 1 + 18 + 5 \bmod 26 = 6$.
- the hash function $h2$, which works the same but skips every second letter (e.g. $h2(hare) = h1(hr)$).

(a) Give the filter table T for a Bloom filter based on D and these two hash functions.

(b) Give one string $m \notin D$ that will be rejected by this filter.
Show that your string is indeed rejected.

Consider the password `authentication`. Note that this password is rejected by the above Bloom filter.

(c) Describe a hash function $h3$ such that a new Bloom filter based on dictionary D and hash functions $h1, h2, h3$ will not reject the password `authentication`. Show that `authentication` is not rejected by this new Bloom filter.

Exercise 10.
(Bloom filters) Consider

- a dictionary $D = \{university, luxembourg, kirchberg, introduction\}$.

Compiled from various sources and editted by Sjouke Mauw and Hugo Jonker. For UL internal use only. ©2007-2013, S. Mauw & H. Jonker.

- the hash function $h1 : \{a, \ldots, z\}^* \rightarrow \mathbb{N}$, which takes the numerical value of each letter (a=1, b=2, etc.) and adds all these numbers together modulo 26. Example: $h1(hare) = 8 + 1 + 18 + 5 \bmod 26 = 6$.
- the hash function $h2$, which works the same but skips every second letter (e.g. $h2(hare) = h1(hr)$).

(a) Give the filter table T for a Bloom filter based on D and these two hash functions.

(b) Give one string $m \notin D$ that will be rejected by this filter.
Show that your string is indeed rejected.

(c) Describe a new hash function $h3$ such that a new Bloom filter based on $D, h1, h2, h3$ will not reject your string m. Show that m is not rejected in this new Bloom filter.

Exercise 11.
(Bloom filters) Consider a Bloom filter with a boolean array of length N and k perfect, independent hash functions $h_1, \ldots, h_k$.

(a) What is the probability that a certain bit b of the array is *not* set to true by a certain hash function for the insertion of one item d_0 into the filter?

(b) What is the probability that a certain bit b is not set by *any* hash function upon insertion of d_0 into the filter?

(c) What is the probability that a certain bit b is not set by any hash functions after insertion of M items?

(d) What is the probability that an item $d_1 \notin D$ is rejected by the Bloom filter with a dictionary D of size M?

Compiled from various sources and editted by Sjouke Mauw and Hugo Jonker. For UL internal use only. ©2007-2013, S. Mauw & H. Jonker.