Computer Security (EDA263 / DIT 641)

Lecture 3: **Passwords**

Erland Jonsson
Department of Computer Science and Engineering
Chalmers University of Technology
Sweden

Bad passwords

- Names (own, wife, child, dog, colleague, car, mistress, etc)
- Numbers that can be related to you (telephone-, car-, birth) or “well-known” numbers, such as e, p, Planck’s constant,....
- Based on any other info that can easily be related to you
- “Popwords” (wizard, gandalf, guatama,...)
- word in dictionary or encyclopedia (Swedish, English, Japanese,...)
- special patters (qwertyui,...)
- none of the above backwards!
- none of the above slightly modified! (i.e. +number, with one big letter, etc)

Good passwords (or at least better!):

1) with small and capital characters
2) with numbers and special characters
3) with at least 8 characters (for UNIX)
4) could be typed easily

- 1)-3) to avoid exhaustive search
- 4) to avoid shoulder surfing
- preferably: random,
  but: hard to memorize in that case

Password Rules

- never reveal your password to anyone!
- do not write it down (in any interpretable way)!
- change it regularly (or at least every now and then...)!
- could be typed fast!

memorizing rules:

- first characters of words in a sentence (Ex. “tiaWcics”)
- combine two small words + extra character: (Ex. “end(pagE”)
- the way to work/auntie Ann/... (Ex. GOPAJOle)

Password Attacks

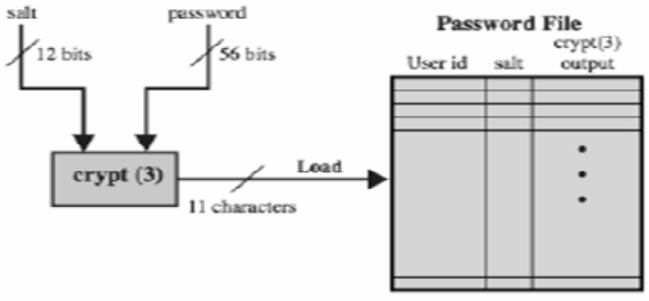
There are three different ways to attack a password:

FIND / GUESS / CRACK

- Find: find note, eavesdrop, keyboard snooping, shoulder surfing, asking for it(!)
- Guess: try “probable” cases, “Joe accounts”
- Crack: exhaustive search, dictionnary attacks
- Example: the UNIX salt feature:

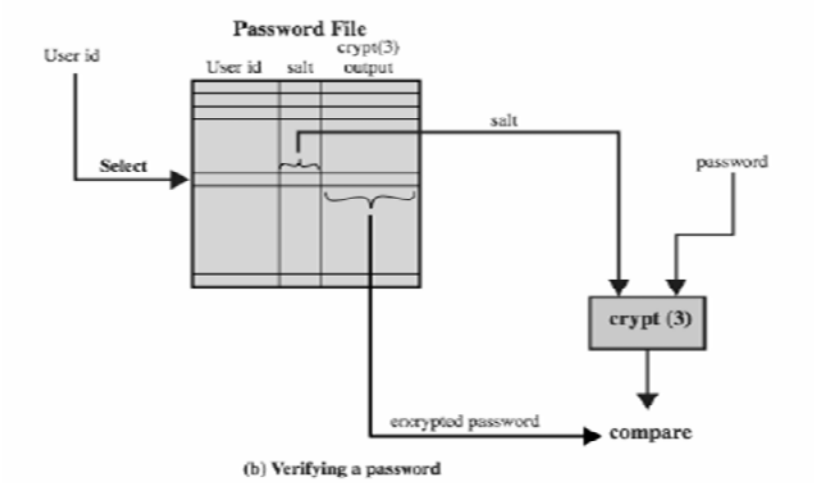
- prevents 2 users with the same passwords from knowing it
- makes exhaustive search for multiple password computationally more expensive

Introducing a new password



(a) Loading a new password

Verifying a password



(b) Verifying a password

UNIX implementation

original scheme
- 8 character password form 56-bit key
- 12-bit salt used to modify DES encryption into a one-way hash function
- 0 value repeatedly encrypted 25 times
- output translated to 11 character sequence

• now regarded as woefully insecure
- e.g. supercomputer, 50 million tests, 80 min

• sometimes still used for compatibility

Improved implementation

- have other, stronger, hash/salt variants
- many systems now use MD5
  - with 48-bit salt
  - password length is unlimited
  - is hashed with 1000 times inner loop
  - produces 128-bit hash
- OpenBSD uses Blowfish block cipher based hash algorithm called Bcrypt
  - uses 128-bit salt to create 192-bit hash value

One-time passwords

A one-time password is a password that is valid only once

- Thus, it is resistant to eavesdropping and wire-tapping.
- One-time passwords can be implemented using special password generators (time-dependent passwords, dynamic password generation) or simply as a list of passwords.
- A special type of one-time passwords are those generated by a challenge-response system.
- In this case the system generates a challenge (seed, nonce), which is different each time and the user calculates a response (=the password) using the challenge. Thus, the password will change every time and can not be re-used.
- In this case, the secret is the *function* that translates the challenge to the response (or: see book Fig 3.11)

Password guessing/cracking (Bruce Schneier)

Password Recovery Toolkit (PRTK) from Access Data

- Password security depends on:
  1) if you can slow down the password testing (in the SW)
  2) the order of guessing by the program
- Guesses 350000 passwords/s (Microsoft OpSys)
- A typical password consists of a root + appendage
- Appendage is a suffix (90%) or prefix (10%)
- PRTK guessing procedure:
  1) dictionary of 1000 pws (e.g. letmein, 123456, etc)
  2) adds 100 common suffixes (e.g 1, 4u, 69, etc)
- => 24 % of all passwords!

Password guessing/cracking cnt'd (Bruce Schneier)

- Exhaustive search of all 4 character strings:
  1) all lowercase, 2) initial uppercase, 3) uppercase
- All with common substitutions (@ for a, 1 for l, etc)
- Collects personal info plus other passwords, which greatly reduces search time

Conclusion: Good passwords are those not found by PRTK

- Forensic Toolkit: scans hard disc for printable strings to create a dictionary => 50 % of passwords
- Windows opsys leaves data (residues) all over the place. May be permanently stored on the hard disc!
- Thus, use opsys insecurity instead of guessing