Protezione e Sicurezza nei Sistemi Operativi:

User Authentication

Ozalp Babaoglu

ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

The Internet Dog

Introduction

- When you first make contact with a service (login, bank, email, social network, etc.) you need to ***identify*** yourself and then ***authenticate*** this identity to prove who you claim to be
- ***Authentication*** is the basis for performing ***Authorization***
- Authentication of humans is different from authentication of messages or services
- Humans are not good at remembering or computing

User Authentication

- Authenticating humans can be based on
  1. Something you know (password, PIN)
  2. Something you have (security token)
  3. Something you do
  4. Something you are (biometrics)
  5. Where you are
- Options 2, 3 and 4 usually require special hardware support
- Option 1 is by far the most common

Password-based authentication

- Leaves no trace of security breaches
- Impossible to prove your innocence if someone misuses your identity
- There is always the possibility that passwords can be guessed
  - Poor appreciation of security among users
  - Short passwords
  - Trivial, easily-guessed passwords
- There is also the possibility that passwords can be "captured"
  - Intruder overlooking your shoulder while entering password
  - Key logger
  - Login spoofing
  - Network sniffing
- Possibility of on-line or off-line attacks

Password-based authentication

- An attacker can always try to guess a password (brute force)
- Let P be the ***probability of successfully guessing*** a password during an ***interval*** of T units of time
- Let G be the ***guess rate*** (number of guesses per unit of time) and N be the ***password space***
- $P \simeq (G \times T) / N$
- General strategies for reducing P:
  - Reduce T — password "aging" — limit validity of passwords
  - Increase N — enforce long, complex passwords
  - Reduce G — artificially slow down the rate of guesses

Password-based authentication

- ***On-line attack***: the system itself is used to verify the correctness of guesses
  - Usually unavoidable if the system has to be physically or remotely accessible
- Defenses:
  - Slow down rate of guesses (decreases G) by inserting a delay between attempts
  - Limit number of incorrect attempts (3 wrong PINs, the phone blocks, Bancomat eats your card)
  - Report date/time/location of last successful login at the next login

Password-based authentication

- ***Off-line attacks***: verify the correctness of password guesses on a system *different* from the one being targeted
- Based on pre-constructed lists of potential passwords
- Need access to passwords in some stored form

Password-based authentication

- How to save passwords
  - As clear text in a file protected by the operating system's access control mechanisms — subject to abuse by privileged users, administrators
- Password encryption
  - Can be based on a *one-way hash function* $f()$
  - The password file contains ***digests*** of the passwords and not the clear text
  - At login, compute the digest of the supplied password by the user and compare it to the value stored in the file
  - Password file in Unix/Linux: **`/etc/passwd`**

```
smithj:Ep6mckrOLChF:561:561:Joe Smith:/home/smithj:/bin/bash
```


Dictionary Attack

- Obtain a copy of the file containing encrypted passwords (***digests***)
- Obtain a file containing list of common words (***dictionary***)
- For each word w in the dictionary, compute its digest using $f(w)$ and compare it to the digests in the password file
- All matching entries correspond to users who have set their password to w
- Can be much more sophisticated by transforming w in common ways (backwards, 2-letter permutations, etc.)
- Can be mechanized through easily-available programs such as **`crack`**

Dictionary Attack

List of common words (dictionary)

```
Achille
Adriano
Africa
Afrodite
Agnese
Agrigento
Alberto
Aldo
Alessandro
Alessio
Ambrogio
America
Amilcare
Anastasia
Ancona
Andrea
Anna
Annibale
Anselmo
Antonino
Antonio
Aosta
...
```

Password file

```
root:ikgjioe9043jb:0:0:...
rossi:wsfl4i4gjio:500:500:...
bianchi:sdiweo38d:501:501:...
franchi:bwjk2lks4df:502:502:...
neri:osdtrkl9dfb:503:503:...
orsi:gi5ikwsdvo:504:504:...
tamburini:lkqweoibve4s:505:505:...
gallo:osdtrkl9dfb:506:506:...
```

$f(\texttt{Achille}) = \texttt{plltuwxkbgp}$

$f(\texttt{Annibale}) = \texttt{osdtrkl9dfb}$

Conclude that users **`neri`** and **`gallo`** are both using the password "**`Annibale`**"

Dictionary Attack

Defenses:

- Limit access to the password file through OS
- Since this is an offline attack, we cannot slow down the guess rate by adding artificial delays between attempts
- But we can artificially slow down the one-way hash function that is used to compute digests (Unix applies DES 25 times to an all-zero block with the password as the key)
- "Salting" of passwords to prevent global attacks
- "Shadow" passwords: separate encrypted passwords from other information typically contained in the password file (e.g., real name of user, office location, telephone number, etc.)

Historical Note

- October 2019, ArsTechnica.com “Forum cracks the vintage passwords of Ken Thompson and other Unix pioneers”
  - Last week, technologist Leah Neukirchen reported finding a source tree for BSD version 3, circa 1980, and successfully cracking passwords of many of computing’s early pioneers. In most of the cases the success was the result of the users choosing easy-to-guess passwords

Historical Note

- Unix co-inventor ***Dennis Ritchie***, for instance, used “`dmac`” (his middle name was MacAlistair); ***Stephen R. Bourne***, creator of the Bourne shell command line interpreter, chose “`bourne`”; ***Eric Schmidt***, an early developer of Unix software and past executive chairman of Google parent company Alphabet, relied on “`wendy!!!`” (the name of his wife); and ***Stuart Feldman***, author of Unix automation tool *make* and the first Fortran compiler, used “`axolotl`” (the name of a Mexican salamander).
- Weakest of all was the password for Unix contributor ***Brian W. Kernighan***: “`/.,/.,`”

Historical Note

- But there were at least five plaintext passwords that remained out of reach. They included those belonging to Turkish computer scientist ***Özalp Babaoğlu***, Unix software developer ***Howard Katseff***, and crucial Unix contributors ***Tom London*** and ***Bob Fabry***. But the uncracked hash that seemed to occupy Neukirchen the longest was the password used by ***Ken Thompson***, another Unix co-inventor
- “I never managed to crack Ken's password with the hash `ZghOT0eRm4U9s`, and I think I enumerated the whole 8 letter lowercase + special symbols key space,” Neukirchen reported on the *Unix Heritage Society* mailing list. “Any help is welcome.”

Historical Note

- It took Nigel Williams 4+ days on an AMD Radeon Vega64 running hashcat at about 930MHz to crack Thompson’s plaintext password: “`p/q2-q4!`” (descriptive notation for a common opening move in the game of Chess)
- A few hours after Williams’ message, Arthur Krewat provided the passwords for the four remaining uncracked hashes:

```
Katseff:  graduat;
Babaoglu: 12ucdort      üç dört
Fabry:    561cml..
London:   ..pnn521
```

Salting

- When user U chooses a password P, the system stores for user U two quantities: S and Q
- S, called the ***salt***, is a random number generated by the system when the user sets her password
- Q is the digest obtained through $f(P \mid S)$ where f is a one-way hash function
- Only S and Q (***not*** P) are stored in the password file along with the user name U

Salting

- When user U wants to authenticate herself to the system, she identifies herself as U and provides her password P:
  - the system reads S and Q associated with user U
  - concatenates S with P and applies f to obtain Q^*
  - compares Q^* with Q
  - if $Q^* = Q$ then authentication succeeds, otherwise it fails
- If an attacker is able to read the password file, it obtains S and Q but is ***not*** able to derive P from them

Salting

- The same password has different encrypted forms (digests) depending on the salt
- Salting of passwords prevents global attacks exploiting the fact that many users use the same password for multiple services or systems
- In Unix, the salt (12 bits long) is used to slightly change the DES internal function (E-Box) and is stored as a 2-character string in the password file

(Lack of) Salting

- The use of a (randomly-generated) salt which is different for each user and each site makes it difficult to obtain the passwords for multiple users or multiple sites simultaneously
- (June 2012) ***LinkedIn*** and ***eHarmony*** don't take the security of their members seriously:

  "… both companies' disastrous password breaches of the past two days, which exposed an estimated 8 million passwords. LinkedIn and eHarmony encrypted, or "hashed" the passwords of registered users, but neither salted the hashes with extra data

(Lack of) Salting

- Why you should always salt your password hashes?

  It's very difficult to reverse a hash, such as by running "5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8" through some sort of formula to produce "password". But no one needs to. If you know that "password" will always result in the SHA-1 hash "5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8", all you have to do is look for the latter in a list of password hashes to know that "password" is a valid password

Unix Shadow Passwords

- In standard Unix, the file **`/etc/passwd`** is readable by everyone because it contains information (name, last name, login shell, home directory, etc.) in addition to passwords
- Which makes it an easy target for dictionary attacks
- More recent versions of Unix implement a *shadow password* mechanism where passwords are removed from **`/etc/passwd`** and are stored in a separate file **`/etc/shadow`**, readable only by root
- Example of a **`/etc/passwd`** file with shadow passwords

```
smithj:x:561:561:Joe Smith:/home/smithj:/bin/bash
mezzina:x:501:501:Leonardo Mezzina:/home/mezzina:/bin/bash
trotter:x:502:503:Guido Trotter:/home/trotter:/bin/bash
hughes:x:503:504:Dino Hughes:/home/hughes:/bin/bash
acerbett:x:504:505:Stefano Acerbetti:/home/acerbett:/bin/bash
```


Advice for system administrators

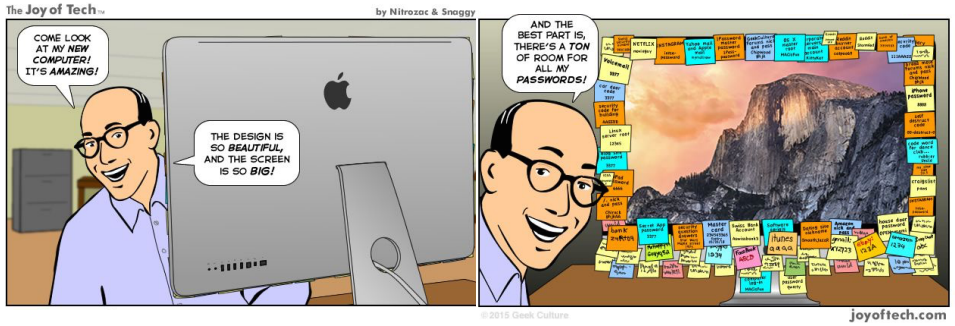
- Always set passwords explicitly and never leave default values
- Educate users on the importance of choosing non trivial passwords
- Periodically run cracking software on own system to reveal presence of weak passwords
- Require remote users to use one-shot passwords or other secure techniques (disable telnet, ftp)

Advice for system administrators

- Force users to choose strong passwords when they create their accounts or change their passwords
  - Impose a minimum password length (at least 8 characters)
  - Require mixed format (at least some non-alpha characters)
  - Reject passwords that can be obtained from simple transformations of common words (dictionary)
- Use "password aging" (must be used within reason)

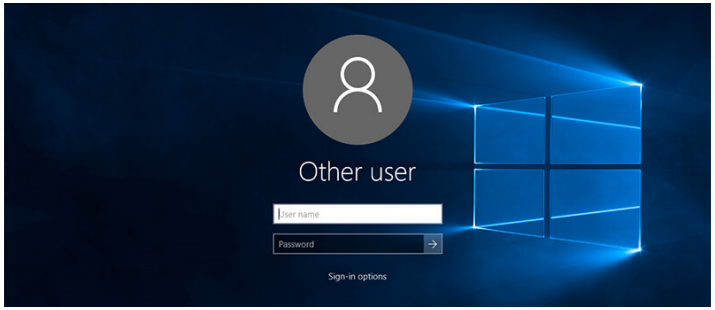
Creating a password

cabbage
Sorry, the password must be more than 8 characters.
boiled cabbage
Sorry, the password must contain 1 numerical character.
1 boiled cabbage
Sorry, the password cannot have blank spaces.
50fuckingboiledcabbages
Sorry, the password must contain at least one upper case character.
50FUCKINGboiledcabbages
Sorry, the password cannot use more than one upper case character consecutively.
50FuckingBoiledCabbagesShovedUpYourArse,IfYouDon'tGiveMeAccessImmediately
Sorry, the password cannot contain punctuation.
NowIAmGettingReallyPissedOff50FuckingBoiledCabbagesShovedUpYourArseIfYouDontGiveMeAccessImmediately



- Typical login mechanisms

```
solaris console login: root
Password:
Login incorrect
solaris console login:
```



- The attacker writes a program (textual or graphic) that generates a ***fake login window*** on the screen
- Waits for the user to login with her credentials
- Capture the login/password pair and either store it locally or send it to a remote site
- On the screen, display “Login incorrect”
- Start the real login program, for example by killing the running shell
- The victim believes to have mistyped her password, and tries again with the real login program (and succeeds)

\

General defenses against login spoofing based on ***mutual authentication***:

- The user authenticates himself to the host
- The host authenticates itself to the user
- Based on cryptographic techniques such as digital signatures and certificates

Phishing

- "Modern" incarnation of login spoofing
- Phishers attempt to fraudulently acquire sensitive information such as passwords and credit card details by masquerading as a trustworthy person or business
- Typically carried out using email or instant messaging, but phone contact has been used as well
- Often relies on social engineering

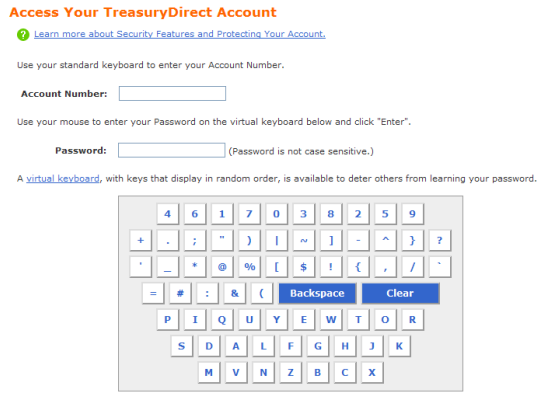
Keyloggers

- ***Keyloggers*** are usually designed as spyware and come in the form of a Trojan horse, can record your passwords, can detect when you type digits checking to see if it's a credit card, bank account or other information you consider private and personal
- Spyware Keyloggers are also used to track your surfing habits
- Keyloggers are usually software but hardware versions also exist

Keyloggers

Keylogger Defenses

- Spyware detection/removal programs
- Firewall for blocking outgoing network traffic
- Virtual keyboards



Packet sniffing

- ***Packet sniffer*** is a piece of software that can analyze traffic on the attached network through a promiscuous interface
- Tries to identify packets containing login/password pairs that are transmitted as plaintext by programs such as `telnet`, `rlogin` or `ftp`
- Stores the captured login/password pair either locally or send it to a remote site for future use

Packet sniffing

- General defenses are based on cryptographic techniques for obfuscating passwords
- Require that the password is never sent in the clear over the network
  - Challenge-response schemes based on symmetric/asymmetric cryptography
  - Challenge can be implicit (such as real time)
- Require that a given password can be used only once
  - "One-time" password schemes such as S/Key

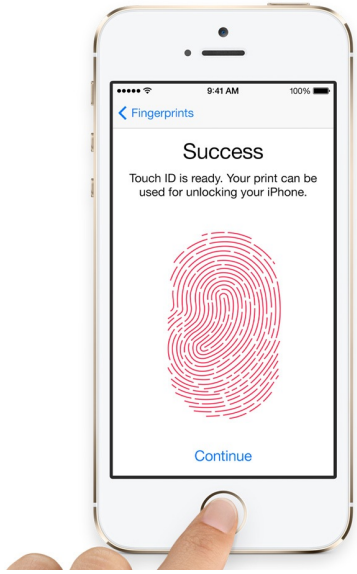
User Authentication based on "something you are"

- Known as "biometrics"
  - Finger print (TouchID)
  - Voice print
  - Retinal patterns
  - Facial features (FaceID)
- Typically require hardware support to acquire
- Chosen biometric should minimize both ***false negatives*** and ***false positives***

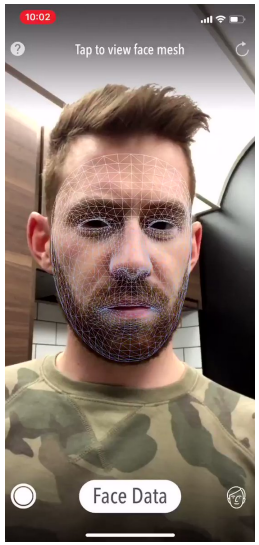
Biometrics

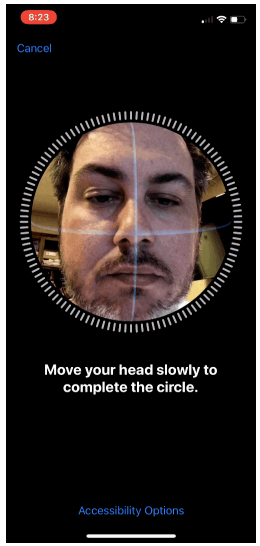
- Desirable properties for a chosen biometric:
- *Universality*: Every person must posses them
- *Uniqueness*: Two different persons must not have the same characteristics
- *Permanence*: Characteristic should not be alterable or change over time
- *Acquirability*: Characteristic easy to acquire and quantify

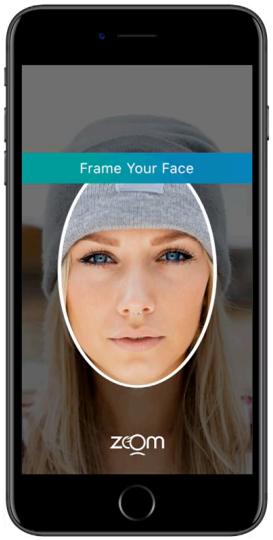
Biometrics — Touch ID



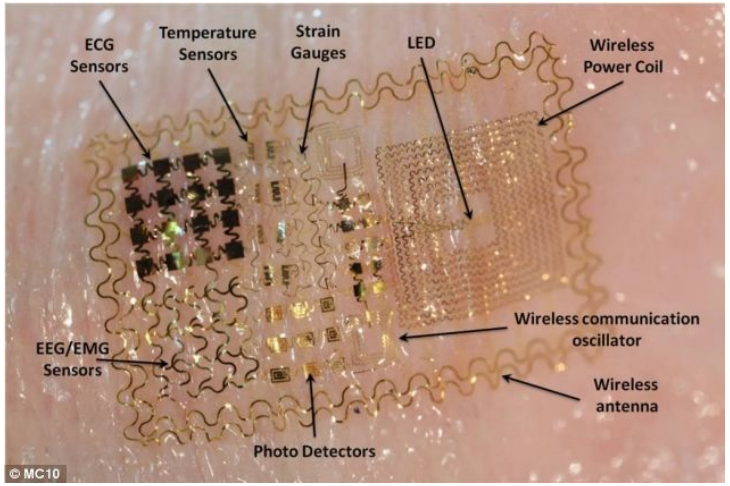
Biometrics — Face ID







Biometrics — RFID “Tattoos”



Biometrics — RFID “Tattoos”

User Authentication based on “something you do”

- Certain human actions can serve to uniquely identify them
- Keystrokes authentication: keystroke intervals, pressure, duration, stroke position (where the key is struck)
- Velocity, acceleration, pressure of pen when writing

Picture Passwords (Windows 8)

Graphical equivalents of passwords

Picture Passwords (Windows 8)