



Security (CS4028)

Lecture 10. Authentication

Chunyan Mu

`chunyan.mu@abdn.ac.uk`

The School of Natural and Computing Sciences

Schedule

	Week	Lecture 1	Lecture 2	Tutorial
	1	Intro to course & security	Intro to Crypto	I
	2	Symmetric Crypto	Hash	Math for crypto
	3	Asymmetric Crypto-1	Asymmetric Crypto-2	Symmetric Crypto
	4	Signatures	Zero Knowledge Proof	Asymmetric Crypto
⇒	5	Certificates	Authentication	Signature & certificates
	6	Access Control	AC models	Authentication
	7	Information flow	Security management	Access control
	8	Protocols	Communications	Concepts & management
	9	Network security	Network security	Protocols and communications
	10	Advanced topics	Advanced topics	Network, web, malware & mgt
	11	Revision		

Outline

Introduction

- why user authentication?
- techniques

By something you have

- passwords
- Cracking passwords
- Salting
- Password Management

By personal characteristics

- Why biometrics
- failure rates

Summary

Outline

Introduction

- why user authentication?

- techniques

By something you have

- passwords

- Cracking passwords

- Salting

- Password Management

By personal characteristics

- Why biometrics

- failure rates

Summary

Objectives

This part will address the following topics

- ▶ Concept of user identification and authentication
- ▶ Main authentication techniques:
 - ▶ something you know:
 - ▶ password, personal information (what you do, where you are ...)
 - ▶ something you possess:
 - ▶ physical key, ticket, passport, token, smart card
 - ▶ what you are (biometrics):
 - ▶ fingerprints, voiceprint, signature dynamics

Identification and Authentication

- ▶ When logging on to a computer/secure area/ATM you enter
 - ▶ identification information: user name and
 - ▶ information needed to verify a claimed identity: password
- ▶ The 1st step is called **identification**:
 - ▶ you announce who you are
- ▶ The 2nd step is called **authentication**:
 - ▶ you prove that you are who you claim to be

Identification and authentication

Main techniques

- ▶ What the claimant knows
 - ▶ Password, personal information
- ▶ What the claimant possesses
 - ▶ Physical key, ticket, passport, token, smart card
- ▶ What the claimant is (biometrics)
 - ▶ Fingerprints, voiceprint, signature dynamics
- ▶ Where the claimant is
 - ▶ Network address, physical location

Why user authentication?

- ▶ To control access to resources and prevent attacks by unauthorised users
- ▶ To record the actions of the users of a computer system
- ▶ To achieve the above, the system have to:
 - ▶ be able to associate access requests and actions with users
 - ▶ be able to identify and distinguish users
 - ▶ be able to associate users with the actions they perform

How does this work?

User account

- ▶ The system maintains a list of user accounts, each is associated with
 - ▶ identifying information: e.g. user name
 - ▶ each program the user runs
- ▶ A user must be identified before entering a computer system by providing the identifying information

How does this work?

Interactions with the computer

- ▶ Some systems provide a command line interface (CLI) called a **shell**:
 - ▶ users interact with the operating system by typing instructions (program names and parameters)
- ▶ Other systems provide a **desktop** and a graphical user interface (GUI):
 - ▶ users interact with the operating system by selecting items from menus or clicking on icons

Modern operating systems provide both!

How does this work?

Interactions with the computer

- ▶ The shell program is associated with the authenticated user's account
- ▶ We therefore can define security requirements in the form of "Alice can read `file.txt` but Bob cannot":
 - ▶ **authorisation**
- ▶ And we can determine which user (Alice or Bob) performed a security-related action
 - ▶ **accountability**

How does this work?



- ▶ Password are usually not strong secrets
- ▶ Computers are bad at recognising people: a computer will not know if an impostor (Alice) enters Bob's user name and password
- ▶ Management: can be expensive



- ▶ Easy to implement, and the most widely used authentication means

Classification of techniques

Identity verification methods can be classified into three types:

- ▶ by something known (e.g. a password or PIN)
- ▶ by something possessed (e.g. a passport or smart card)
- ▶ by physical characteristic (e.g. a fingerprint, facial characteristics, a signature)

Multifactor authentication (MFA)

- ▶ MFA typically involves more than one category of authentication factors
- ▶ Benefits: enhanced security, fraud prevention ...
- ▶ Common use: Online Banking, Corporate Systems, Social Media ...

Outline

Introduction

why user authentication?

techniques

By something you have

passwords

Cracking passwords

Salting

Password Management

By personal characteristics

Why biometrics

failure rates

Summary

By something you have

- ▶ Examples of verification schemes of this type
 - ▶ passwords and PINs used by human/human or human/computer
- ▶ Security procedures for passwords
 - ▶ passwords should be issued to individuals and the use of the password can be made accountable to one user
 - ▶ passwords should not be written down
 - ▶ passwords should be chosen to be hard to guess
- ▶ Alternative: list of one-time passwords (use the password exactly once!)

Passwords

Passwords storage

- ▶ Password is not stored in the system:
 - ▶ if unencrypted then readable by systems staff
- ▶ Usual solution - using a one-way function to hide it
 - ▶ easy to compute, but difficult to invert
- ▶ For each user, system stores (user name, $f(\text{password})$), where f is a transformation function (e.g., one-way cipher) in a password file
 - ▶ $f(\text{password})$ is easy to compute
 - ▶ from $f(\text{password})$, password is difficult to compute
- ▶ When user enters the password, system computes $f(\text{password})$; match provides proof of identity

Password abuse

- ▶ Many (most?) users (for obvious reasons):
 - ▶ choose poor passwords
 - ▶ do not maintain the secrecy of their passwords
- ▶ Users often:
 - ▶ write their password down
 - ▶ tell other people their password
 - ▶ log on to their machine and then leave it unattended
- ▶ Training and security awareness are very important here!

Vulnerabilities on passwords

- ▶ Inherent vulnerabilities
 - ▶ Easy to guess or snoop
 - ▶ No control on sharing
- ▶ Practical vulnerabilities
 - ▶ Visible if unencrypted in distributed and network environment
 - ▶ Susceptible for replay attacks if encrypted naively
- ▶ Password advantage
 - ▶ Easy to modify compromised password.

Cracking passwords

The password space

- Try to make a back-of-the-envelope, conservative estimate about what can be protected against brute-force.
 - (a) $2^8 (= 256)$ characters in extended ASCII.
 - (b) Typically fewer than 2^7 immediate on keyboard. Be conservative, so say $2^6 (= 64)$.

pwd length (exact)	Num pwds (a)	Num pwds (b)
6	$(2^8)^6 = 2^{48}$	$(2^6)^6 = 2^{36}$
8	$(2^8)^8 = 2^{64}$	$(2^6)^8 = 2^{48}$
12	$(2^8)^{12} = 2^{96}$	$(2^6)^{12} = 2^{72}$
16	$(2^8)^{16} = 2^{128}$	$(2^6)^{16} = 2^{96}$

Cracking passwords

The password space

- ▶ Make conservative estimate about what can be protected. Therefore slightly overestimate what attacker can do.
- ▶ On a single CPU, *John the Ripper* can do 90,000,000 (just under 2^{27}) crack attempts per second for many standard hash algorithms. (10^{10} or 2^{34} on a GPU)
- ▶ A bit under 2^{17} secs per day, and 2^{25} per year.
- ▶ This won't always be possible for every hash etc.

time	Num guesses (CPU)	GPU
1 sec	2^{27}	2^{34}
1 day (2^{17} sec's)	$2^{27} \times 2^{17} = 2^{44}$	2^{51}
1 year (2^{25} sec's)	$2^{27} \times 2^{25} = 2^{52}$	2^{59}

Cracking passwords: the password space

- ▶ Maybe you are a “well-resourced attacker” and have a million (say 2^{20}) GPUs. Then you can still only try $2^{59} \times 2^{20} = 2^{79}$ in a year.
- ▶ Of course, on average, you only half to search half the space before you find a particular password.
- ▶ On top of this, a typical hash has a number of collisions - several passwords can give the same hash as a real password (each of these will then pass as authentic).
- ▶ Others have estimated a few years ago that a search space of size 2^{80} was likely not to be feasible (for encryption rather than hashing).
 - ▶ Brute force of 2^{56} was done in 1998 by “Deep Crack” for the DES cipher.
 - ▶ The underlying physics (energy consumption) mean that it is highly unlikely that more than 2^{128} can be done.
 - ▶ But there may be quantum search techniques ...
- ▶ What other techniques can be used to try to improve the search?
- ▶ Also, are you searching for one particular password or any?
- ▶ In practice, many of the possible passwords don't get used much.

Dictionary attacks

- ▶ Dictionary attack: guessing password by trial and error.
 - ▶ Usually using a wordlist of common words to tun search.
 - ▶ Both on-line (type 1) and off-line (type 2) versions occur.
- ▶ **Offline** (type 2)
 - ▶ Situation: Password file obtained by attacker. Hash function inferred.
 - ▶ Method: hash test authentication data until you find something that produces a match with something in the complementary data. Use the wordlist to speed up.
 - ▶ Use other techniques like *lookup tables*, *hash chaining* and *rainbow tables*.
 - ▶ There are attack tools for this, e.g., *John the Ripper*
- ▶ **Online** (type 1)
 - ▶ Situation: Password file not available or usable.
 - ▶ Method: Guess directly. That is, try to authenticate directly.
 - ▶ May result in lock-out, or discovery.
 - ▶ There are attack tools for this, e.g., *Hydra*

Dictionary attacks on passwords

Attack 1

- ▶ Create dictionary of common words and names and their simple transformations
- ▶ Use these to guess password

Attack 2

- ▶ Usually f is public and so is the password file (encrypted)
- ▶ Compute $f(\text{word})$ for each word in dictionary
- ▶ Find match

Attack 3

- ▶ Pre-compute dictionary
- ▶ Look up matches

Dictionary attacks

Example

- ▶ Attackers get UNIX password file
- ▶ Hash function is known.
- ▶ Use this as in Type 1 (online). There are programs that automate the process.
- ▶ However, need to gain initial access to get the password file.
- ▶ Often starts with a Type 2 (offline) attack.
- ▶ Guess passwords directly, e.g., using default accounts and passwords

Passwords

Salting

- ▶ Salting is a technique for further protecting passwords.
- ▶ Append additional information s , the **salt**, to each user's password p .
 - ▶ Concatenate (join) the password and the salt $p::s$.
 - ▶ Apply hash function H to $p :: s$, to give a hash value

$$h = H(p :: s)$$

- ▶ Store the pair (h, s) on the system.
- ▶ Rule: no two users get the same salt.
- ▶ The salt is usually generated to be (pseudo-)random

Passwords: Salting

- ▶ If two users have the same password, p , then they will still have different stored complementary data
 - ▶ assuming reasonable salting and hashing.
 - ▶ They get $H(p :: s1)$ and $H(p :: s2)$ for different salts.
- ▶ Salting gives a tiny bit of help with the problem of users re-using obvious passwords between different authentication systems (e.g., for different websites), since the stored hashes for each will be different, so it will not be obvious to an attacker that they are identical.
- ▶ The main advantage of salting is that it slows down attacks on a whole list of passwords.
 - ▶ It makes it impossible to search for passwords of several users simultaneously.
 - ▶ We typically assume that attacker can get the pairs (h, s) and hash function H .
 - ▶ Further explained on the next slide.

Passwords

Why salting works?

Attack on system without salts and m passwords considered as good guesses (the wordlist/dictionary):

- ▶ Try guess p_1 ; compute $h^1 = H(p_1)$; check entire password file to see if h^1 present;
 - ▶ A longer password file means more opportunities for a match.
- ▶ Try another guess $h^2 = H(p_2)$
- ▶ Do this for all m passwords: p_1, p_2, \dots, p_m .
- ▶ There are m computations involved in trying to crack the password file.

Passwords

Why salting works?

Attack with n known salts s_1, \dots, s_n and m passwords considered as good guesses (the wordlist/dictionary):

- ▶ Different salt goes with each password
- ▶ For a pair (h_1, s_1) , try guess p_1 with salt s_1 ; compute $H(p_1 :: s_1) = h^{1,1}$; is $h^{1,1} = h^1$?
- ▶ If that fails, then guess p_2 with salt s_1 , is $H(p_2 :: s_1) = h^{2,1} = h^1$?
- ▶ Etc., all the way through to the guess p_m with salt s_1
- ▶ For a different pair (h_2, s_2) , now have to compute $H(p_1 :: s_2), \dots, H(p_m :: s_2)$.
- ▶ Do this for all n salts. There are m passwords tried with each salt.
- ▶ **There are now $n \times m$ computations in the crack.**

With salting, we can't re-use the same test hash value across multiple passwords

- ▶ e.g., using $h^{1,1}$ to try to crack (h_2, s_2) is very unlikely to work.

Passwords

Why salting works?

- ▶ Salting gives a kind of herd-level protection.
 - ▶ Less attractive for attacker.
 - ▶ Better defence for whole herd.
 - ▶ Better defence for owner of system hosting passwords.
- ▶ Salting makes no difference to cracking a particular individual password in an offline attack.

Passwords

Lookup and Rainbow tables

- ▶ **Lookup table**: attacker has a table of pre-computed hashes for known, common passwords.
- ▶ Just search through the lookup table and the password file until a matching hash is found. Then pick out the corresponding password.
- ▶ **Rainbow tables** are a clever probabilistic modification of this technique.
- ▶ Salting also helps to defeat lookup and rainbow tables
 - ▶ Table needs to be too big.

Passwords

Insecure storage on system and caching

- ▶ **Password caching**: in reality password is stored in intermediate locations like buffers, caches, web-pages.
 - ▶ Management not under user control. User may not know where or when the password persists.
- ▶ Early online banking examples, where cookies stored information about user to allow them to move backwards through web-pages. Closing the banking application but not killing the browser allowed a second user to access the first user's account.
- ▶ This is an example of a quite low-level implementation flaw, that may compromise security policies that are typically stated at a higher level of abstraction.

Passwords

Further password vulnerabilities

More Vulnerabilities and Attacks on Password/PIN systems where User Plays a Role.

- ▶ Guessing attacks
 - ▶ including “spouse” attacks
 - ▶ Use of guessable information about the user.
- ▶ Shoulder-surfing attacks
- ▶ Skimming attacks (e.g. at ATMs)
- ▶ Sniffing attacks (key loggers, click-and-clack attacks)
- ▶ Spoofing
- ▶ Password re-use

Passwords

Spoofing

- ▶ Authentication, as above, is asymmetric.
 - ▶ This is in contrast to mutual authentication scenarios.
- ▶ We can use this fact to get the user to send the password to the attacker:
 - ▶ Trick user into trying to authenticate to a compromised end-point (e.g., a fake website).
 - ▶ Password is then sent to wrong place, where the attacker may capture it.
 - ▶ Authentication may appear to the user to abort or fail, and afterwards the user is directed back to the true log-in.
 - ▶ A variant would be to pass the user automatically, so that they do actually log-in.
- ▶ Phishing and social engineering exploits have some similar characteristics.

Passwords

Spoofing prevention

- ▶ Display to user the number of failed log-ins.
 - ▶ If first log-in fails, and you are then told at the second attempt that this is your first attempt, then you should be suspicious.
- ▶ **Mutual authentication:**
 - ▶ e.g., in a distributed system, the system could be required to authenticate itself to the user.
- ▶ **Trusted path:** guarantee that the user communicates with the true other party (the guard) and not with a spoofing program.
 - ▶ Authenticated connection, and providing confidentiality (secrecy) and integrity.
 - ▶ On a local OS, the guard is often called the **security kernel** or **reference monitor**.
 - ▶ In some Windows versions it is better to do CTRL-ALT-DEL to get the **secure attention sequence**, rather than use another already-displayed logon screen.

Password Management

Password Construction Policy

- ▶ Organisations set password policies
- ▶ These state how passwords must be constructed (usually to satisfy and implemented checking mechanism):
 - ▶ Password length
 - ▶ Password content
 - ▶ Frequency of change (password ageing)
 - ▶ Number of login attempts; re-setting
- ▶ **Password space:** the set A of all possible passwords.
 - ▶ i.e., the set of all sequences of characters that constitute allowable passwords.

Password Management

The generalised Anderson Formula

- ▶ Let
 - ▶ P be the probability that attacker guesses a password in a specified time period
 - ▶ G be the number of guesses that can be tested in one time unit.
 - ▶ T be the number of time units during which guessing occurs
 - ▶ N be the number of all possible passwords

- ▶ Then:

$$P \geq TG/N.$$

- ▶ This gives a lower bound for P (a sort of probabilistic best case for the defender) assuming an attacker who just guesses at random.
- ▶ We'd really like an upper bound too.

Password Management

Example: the generalised Anderson Formula

► Set-up:

- Pwd chars from alphabet with 96 characters ($A = 96$)
- No other constraint on strings except length; our system will allow all passwords no greater than a certain length S .
- 10^4 guesses can be tested per second.
- Security engineer wants a probability of no greater than a half that there is a successful guess in 365 days.

► **Problem synopsis:** Find *necessary* condition on S to give $0.5 \geq P$.

► By Anderson's formula ($P \geq TG/N$), we must have:

$$N \geq TG/P = (365 \times 24 \times 60 \times 60) \times 10^4 / 0.5 \approx 6.31 \times 10^{11}.$$

- There are $N = \sum_{i=0}^S A^i$ passwords of length less than or equal to S .
 - because there are $A^S = A \times A \times \dots \times A$ pwds of length S .
- So we need S such that $\sum_{i=0}^S 96^i \geq 6.31 \times 10^{11}$.
- Calculation shows that we need $S \geq 6$.

Tokens: something you have

Time-Based Token Authentication

Login: mcollings

Passcode: 2468159759

PASSCODE = PIN + TOKENCODE

Token code:
Changes every
60 seconds



Clock
synchronized to
UCT

Unique seed

From Security in Computing, Fifth Edition, by Charles P. Pfleeger, et al

Outline

Introduction

- why user authentication?
- techniques

By something you have

- passwords
- Cracking passwords
- Salting
- Password Management

By personal characteristics

- Why biometrics
- failure rates

Summary

By personal characteristics

Why personal characteristics?

- ▶ Passwords may be revealed or guessed
- ▶ Tokens may be lost or stolen
- ▶ Hence use of personal characteristics may be harder to forge
- ▶ Device measuring characteristics must be trusted (e.g. physically secure) otherwise replay possible

By personal characteristics

What is biometrics?

- ▶ Biometrics is the measurement and statistical analysis of biological data
- ▶ In IT, biometrics refers to technologies for measuring and analysing human body characteristics for **authentication** and/or **identification purposes**.
- ▶ Definition by Biometrics Consortium
 - ▶ **automatically recognising a person using distinguishing traits.**

By personal characteristics

How does it work?

Each person is unique!

- ▶ what are the distinguishing traits that make each person unique?
- ▶ how can these traits be measured?
- ▶ how different are the measurements of these distinguishing traits for different people?

Verification vs. Identification

Two basic modes of operation of a biometrics system

- ▶ Verification (one-to-one comparison) confirms a claimed identity:
 - ▶ claim identity using name, user id, ...
- ▶ Identification (one-to-many comparison) establishes the identity of subject from a set of enrolled persons, e.g.
 - ▶ employee of a company?
 - ▶ member of a club?
 - ▶ criminal in forensics database?

Verification vs. Identification

Two basic modes of operation of a biometrics system

- ▶ Verification (one-to-one comparison) confirms a claimed identity:
 - ▶ claim identity using name, user id, ...
- ▶ Identification (one-to-many comparison) establishes the identity of subject from a set of enrolled persons, e.g.
 - ▶ employee of a company?
 - ▶ member of a club?
 - ▶ criminal in forensics database?

Practical questions

- ▶ is the claimant the owner of this card/computer/document? - **verification**
- ▶ should this individual be given access to the system/room/file? - **identification**

Biometric identifiers

Properties of biometric identifiers

An ideal identifier (the distinguishing traits) should possess the following properties:

- ▶ Universality
- ▶ Uniqueness
- ▶ Stability
- ▶ Collectability
- ▶ Performance
- ▶ Acceptability
- ▶ Forgery resistance

Biometric technologies

Biometric identification schemes

Many different automatic identification schemes have been proposed and used, including:

- ▶ recognition of signatures
- ▶ fingerprint analysis
- ▶ voice recognition
- ▶ retinal scan
- ▶ iris scan
- ▶ face recognition
- ▶ hand geometry

Failure rates

The matching module rates the similarity between the collected biometric data and the reference template.

- ▶ If the match score is above a tolerance (or confidence) threshold, the claimant is accepted.
- ▶ If it is below the tolerance threshold, the claimant is rejected.

Failure rates

The matching module rates the similarity between the collected biometric data and the reference template.

- ▶ If the match score is above a tolerance (or confidence) threshold, the claimant is accepted.
- ▶ If it is below the tolerance threshold, the claimant is rejected.

Possible decision outcomes

- ▶ A genuine individual is accepted
- ▶ A genuine individual is rejected (Type I error)
- ▶ An impostor is rejected
- ▶ An impostor is accepted (Type II error)

Failure rates

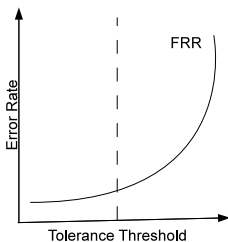
Errors

- ▶ Balance needed between 2 types of error:
 - ▶ Type I: system fails to recognise valid user (**false non-match** or **false rejection**)
 - ▶ Type II: system accepts impostor (**false non-match** or **false acceptance**)
- ▶ There is a trade off between these two types of errors that depends on the security requirements of the application.

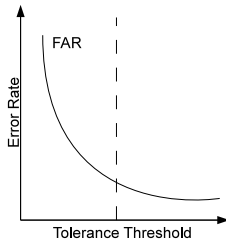
Tolerance threshold

- ▶ Error tolerance threshold is crucial and application dependent:
 - ▶ by setting the threshold for the matching algorithm, we can trade off between the **false acceptance rate (FAR)** and the **false rejection rate (FRR)**
- ▶ Tolerance too small causes Type II errors (impostors admitted)
- ▶ Tolerance too large causes Type I errors (legitimate users rejected)
- ▶ **Equal error rate (EER)**: false non-match (FRR) = false match (FAR).
 - ▶ It is usually used as a criterion to compare different biometric systems

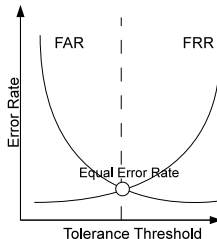
FAR, FRR error curve



Tolerance high:
large of legitimate users rejected



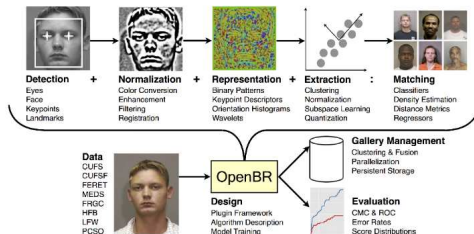
Tolerance low:
large number of impostors admitted



If the tolerance threshold is high,
then a large number of legitimate
users will be rejected

If the tolerance threshold is low,
then a large number of impostors
will be admitted

Implementation (examples)



- ▶ **OpenBR: Open Source Biometric Recognition**
 - ▶ <http://openbiometrics.org/>
 - ▶ A communal biometrics framework supporting the development of open algorithms and reproducible evaluations.
- ▶ **Openface**
 - ▶ <https://cmusatyalab.github.io/openface>
- ▶ **Visa Developers**
 - ▶ <https://developer.visa.com/capabilities/biometrics>

Outline

Introduction

- why user authentication?
- techniques

By something you have

- passwords
- Cracking passwords
- Salting
- Password Management

By personal characteristics

- Why biometrics
- failure rates

Summary

Summary

Identification and authentication

- ▶ Concept of user identification and authentication
- ▶ By something you have: password, tokens
- ▶ By personal characteristics: biometric method

Next week..

- ▶ Access control
- ▶ Security models