1495 UNIVERSITY OF ABERDEEN

CELEBRATING 525 YEARS 1495 – 2020

ABERDEEN 2040

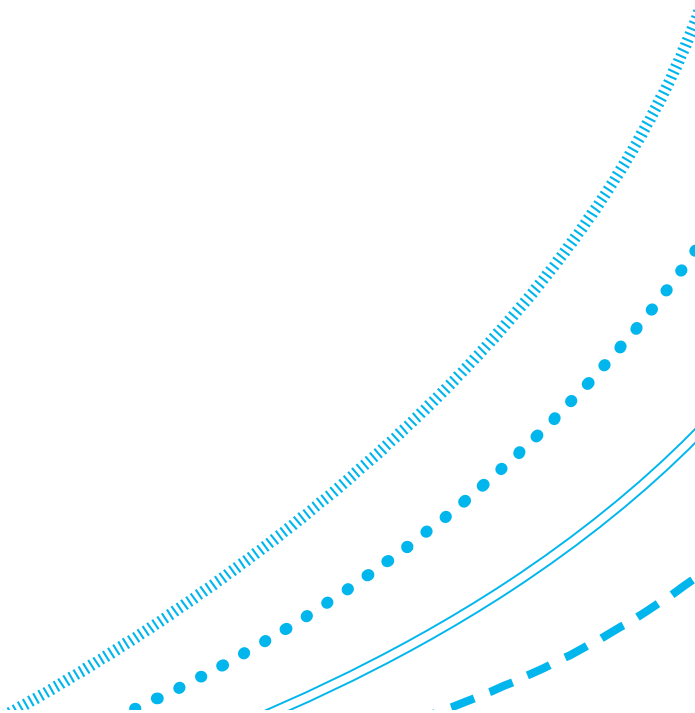
Network Security Technology

Passwords

September 2025

Outline of lecture

1. Password usage, assumptions and attack vectors.
2. Protecting passwords.
3. Cracking passwords.
4. Salting.
5. Further vulnerabilities.
6. Choosing passwords.
7. Managing passwords.
8. Biometrics.



Password authentication

1. Usually two steps:
    1. Identification:
        1. By supplying username.
    2. Verification:
        1. Using a secret agreed in advance – the password.
2. Example:

```
Username: abdnuser123
Password: ************
```

Complementary data for passwords

1. Complementary data is held in a data store. Very often this is a password file.
2. The complementary data must be something that we can quickly verify a password against. It may not be exactly the password itself.
3. Password must be chosen to be something sufficiently secret:
    1. Must also be communicated securely initially.
4. Both the system and the user can keep the password secret:
    1. the system must have security around the complementary data and places where it is used.
    2. the user must have security around the password and take care that it does not leak.
5. These assumptions are often incorrect.

General attack vectors on passwords

1. System-side:
    1. Default passwords, guess passwords with brute force (try them all)
    2. Guess passwords with dictionary attack:
        1. The user may have a role in how hard this is.
    3. Steal complementary data directly: e.g. password file:
        1. If weak use and storage of received authentication data by system. Try to crack passwords.
2. User side:
    1. Copy knowledge (e.g. if found written-down), snooping (shoulder-surfing, key-logging etc.), social engineering, sharing, spoofing.
3. Communications channel: eavesdrop password.
4. Other: e.g. inference from password re-use elsewhere.
5. Many attacks are a mix.

Protecting passwords: one or many

1. User may only care about protecting their own password.
    1. Attacker may only care about getting that particular password.
2. System managers care about protecting all user passwords.
    1. Attacker may only care about finding some, or any one, of the passwords, not a particular one.
3. There are two quite different problems here.

Options for protecting password files

1. Use access controls to password file:
    1. Only privileged users get **write** access to password file.
        1. Otherwise, attacker could change it, and then use the modified data to authenticate.
    2. **Read** protection can be used along with encryption:
        1. *e.g., shadow password files* in UNIX not publicly accessible.
        2. Read protection alone is usually not enough:
            1. Sophisticated attackers will find the file anyway.
            2. `No security through obscurity'
    3. We are using one access control system to mediate access to (the password files used by) an authentication system that in turn is used by an access control system (often the same one).

Options for protecting password files (cont'd)

2. Use cryptographic mechanisms to protect the file contents.
3. Usually, we use both of these things together. The cryptographic mechanisms give a back-up security feature when attackers can work around the access controls.

Complementary data: option 1

1. Store passwords **in the clear.**
2. *This is usually a very bad idea:*
    1. These files are major targets for attackers.
    2. History shows that they may well be compromised by an attacker.
    3. If they are, then the attacker can impersonate legitimate users.

Complementary data: option 2

1. Encrypt the entire password file:
    1. Have to decrypt the entire file every time a verification needs to be performed.
    2. All passwords are then exposed.
    3. Have to encrypt whole file every time a password needs to be changed, or a new user enrolled.
2. If the encryption could be cracked, then all of the plaintext passwords would be compromised.

Complementary data: option 3

1. Encrypt the individual passwords next to the user identities.
    1. Joebloggs2016 : f3a466ee2f11bb5c
2. Has been used in the past.
    1. User password is still exposed during use when inside the system.
    2. An attacker may be able to discover it when in use.
    3. Again, if the attacker can figure out how to decrypt then they can discover all the passwords.

Complementary data: option 4

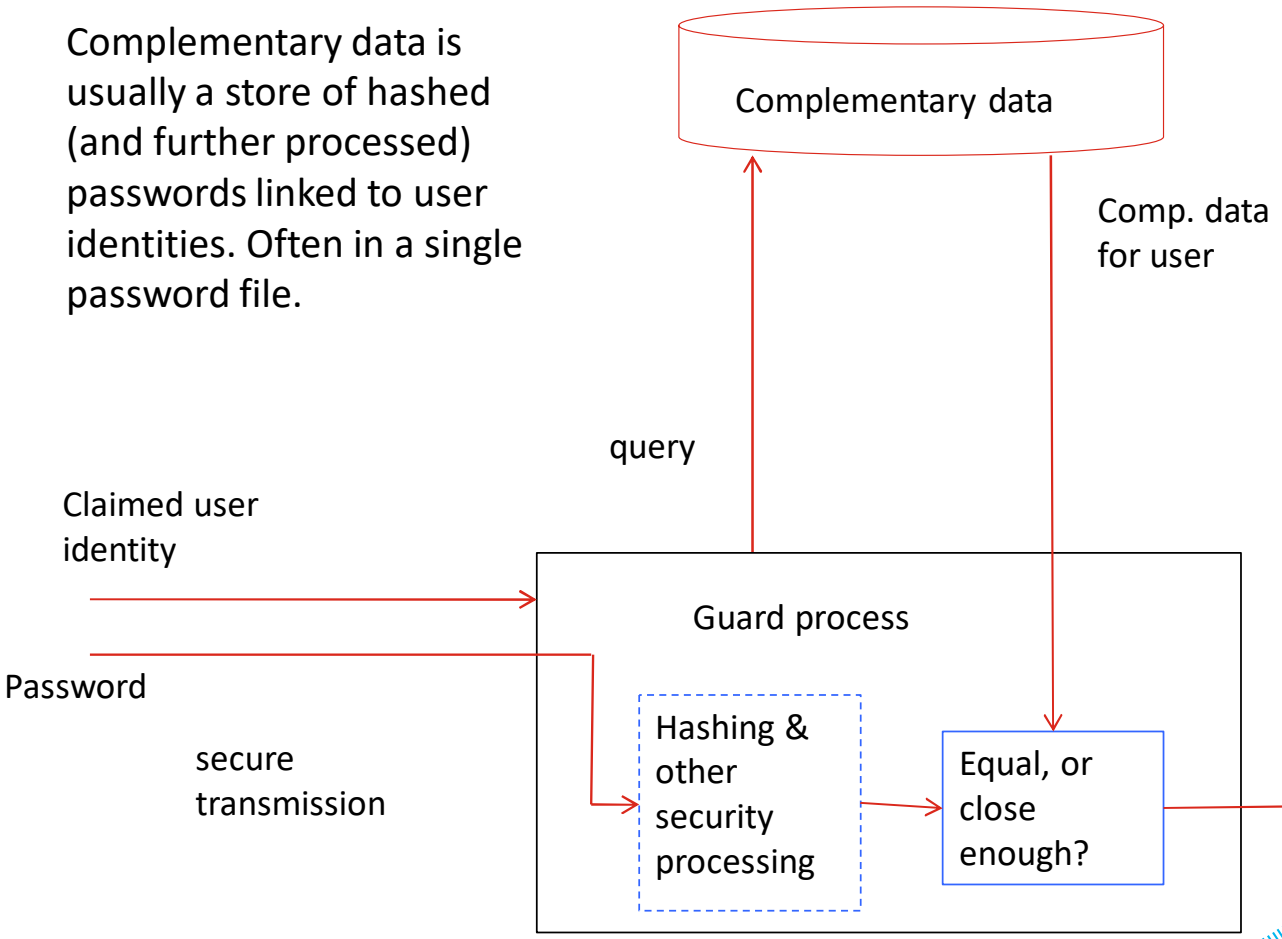
1. Usual solution: have the complementary function be a **cryptographic hash function** that scrambles the authentication information.
    1. This makes it hard to reverse the complementary information into the authentication information.
    2. So even if the attacker gets a hashed password, they can't get the original password, and so can't use it.
    3. Might be slightly easier if they get the whole password file, if they just need to extract any password (rather than a particular one).

Encrypting vs. hashing passwords

1. Arguments for hashing:
    1. no real need to be able to decrypt passwords;
    2. remove risk of lost encryption keys;
    3. ability to produce a hash is sufficient assurance.
2. Arguments against hashing:
    1. introduces pseudo-passwords, i.e. plaintexts that are not passwords for any user but hash to the same value as a genuine password;
    2. attacks to break hashing algorithms are being regularly discovered.
3. The balance of opinion is generally that hashing is the best option.

Typical workflow for protecting passwords

Complementary data is usually a store of hashed (and further processed) passwords linked to user identities. Often in a single password file.



Protection of passwords in UNIX-like systems

1. **Originally:**
    1. Password file contained passwords of users in the clear;
    2. protected against being written or read.
2. **Next:**
    1. salted, encrypted passwords. Later salted, hashed passwords.
    2. The password file (/etc/passwd) is readable by everybody (often), but only superuser can modify. Entries like:

        Name:*Password*: *UserID*:*PrincipleGroup*:*Gecos*: *HomeDirectory*:*Shell*

3. **Modern:** hashed passwords stored in shadow password files (e.g etc/shadow , /etc/master.passwd) that only accounts with special privileges (e.g. root) can read. (Inside /var/db/shadow directory in OS X).

Password protection in Windows

1. (Warning: everything may have changed in Windows 10).
2. Windows Security Account Manager (SAM).
3. At least since XP.
4. Stores hashed passwords in database.
5. Until quite recently this was in:

   `\windows\system32\config .`

   1. May now have changed in recent windows versions.

Cracking passwords

1. Try to make a back-of-the-envelope, conservative estimate about what can be protected against brute-force.
    1. 2^8 (=256) characters in extended ASCII.
    2. Typically, fewer than 2^7 immediate on keyboard.
    3. 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	$(2^8)^{16} = 2^{128}$	$(2^6)^{16} = 2^{96}$

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 000000 (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. However, 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}

The password space (cont'd)

1. 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.
2. Of course, on average, you only half to search half the space before you find a particular password.
3. 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).
4. 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).
    1. Brute force of 2^{56} was done in 1998 by `Deep Crack' for the DES cipher.
    2. The underlying physics (energy consumption) mean that it is highly unlikely that more than 2^{128} can be done.
    3. But there may be quantum search techniques …

The password space (cont'd)

5. What other techniques can be used to try to improve the search?
6. Also, are you searching for one particular password or any?
7. In practice, many of the possible passwords don't get used much.

Default passwords

1. Many systems supplied with default credentials. (e.g. an admin account with a blank or standard password):
    1. admin: admin, guest: guest etc.
    2. PIN for mobile/cell-phone
    3. Password for mobile-phone data networks
    4. Router
    5. Server
    6. `Helpful' sites, e.g. http://www.phenoelit.org/
    7. Big problem if not changed by admins and users.
    8. Penetration testing will show up some of this.
    9. Need to train and incentivize admins and users.

Dictionary attacks

1. Dictionary attack: guessing password by trial and error.
    1. Usually using a wordlist of common words to tune search.
    2. Both on-line (type 1) and off-line (type 2) versions occur.
2. **Offline** (type 2):
    1. Situation: Password file obtained by attacker. Hash function inferred.
    2. 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.
    3. Use other techniques like *lookup tables*, *hash chaining* and *rainbow tables*.
    4. There are attack tools for this, e.g. *John the Ripper*

Dictionary attacks (cont'd)

3. **Online** (type 1):
    1. Situation: Password file not available or usable.
    2. Method: Guess directly. That is, try to authenticate directly.
    3. May result in lock-out, or discovery.
    4. There are attack tools for this, e.g. *Hydra*

Dictionary attack example

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

Salting

1. Salting is a technique for further protecting passwords.
2. Append additional information *s*, the **salt**, to each user's password *p*.
    1. Concatenate (join) the password and the salt *p*::*s*.
    2. Apply hash function *H* to *p*::*s*, to give a hash value *h* = *H*(*p*::*s*)
    3. Store the pair (*h*, *s*) on the system.
    4. Rule: no two users get the same salt.
    5. The salt is usually generated to be (pseudo-)random.

Salting (cont'd)

1. If two users have the same password, *p*, then they will still have different stored complementary data
    1. Assuming reasonable salting and hashing.
    2. They get $H(p::s_1)$ and $H(p::s_2)$ for different salts.
2. 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).
3. Since the stored hashes for each will be different, so it will not be obvious to an attacker that they are identical.

Advantage of salting

1. The main advantage of salting is that it slows down attacks on a whole list of passwords.
    1. It makes it impossible to search for passwords of several users simultaneously.
    2. We typically assume that attacker can get the pairs (*h*, *s*) and hash function *H*.
    3. Further explained on the next slide.

Attacks on systems without salts

1. Attack on system without salts and *m* passwords considered as good guesses (the wordlist/dictionary):
    1. Try guess p_1; compute $h^1 = H(p_1)$; check entire password file to see if h^1 present;
        1. A longer password file means more opportunities for a match.
    2. Try another guess $h^2 = H(p_2)$
    3. Do this for all m passwords: p_1 , p_2 , …, p_m .
    4. *There are m computations involved in trying to crack the password file.*

Attacks on systems with *n* known salts

1. Attack with n known salts s_1 ,…, s_n and *m* passwords considered as good guesses:
    1. Different salt goes with each password *a*
    2. 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 $h^{1,1}=h_1$?
    3. If that fails, then guess p_2 with salt s_1 , is $H(p_{2:}::s_1) = h^{2,1} = h_1$?
    4. Etc., all the way through to the guess p_m with salt s_1
        1. Still the same salt, s_1 .
    5. For a different pair (h_2, s_2), now have to compute $H(p_{1:}::s_2)$, … , $H(p_m::s_2)$.
    6. Do this for all *n* salts. There are *m* passwords tried with each salt.
    7. *There are now n × m computations in the crack* .
2. With salting, we can't re-use the same test hash value across multiple passwords
    1. E.g. Using $h^{1,1}$ to try to crack (h_2, s_2) is very unlikely to work.

Remarks on salting

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

Lookup and rainbow tables

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

Salting: been around for years, but not always used

Name	Year attack/ year acknowledged	Number of Accounts Possibly Affected	Hash/Encryption	Salt	Further information
Dropbox	2012/ 2016	70 000 000	bcrypt for some. Others?	Probably	Dropbox employee re-used password on another site.
LinkedIn	2012- ...	100 000 000 +	SHA-1	No ✗	Seemingly repeated attacks.
Yahoo	2014/ 2016	500 000 000	MD5 claimed for cracked passwords ✗ Company says bcrypt for most.	No for some ✗ and yes for some	Unencrypted answers to password reset questions lost!

Insecure storage on system and caching

1. **Password caching**: in reality, a password is stored in intermediate locations like buffers, caches, web-pages.
    1. Management not under user control. User may not know where or when the password persists.
2. Early online banking examples, where cookies stored information about user to allow them to move backwards through web-pages.
    1. Closing the banking application but not killing the browser allowed a second user to access the first user's account.
    2. 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.

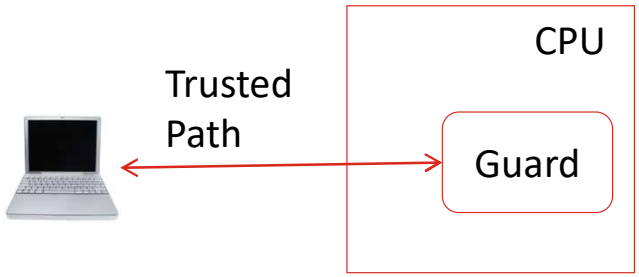
Attacks on password/pin systems where users play a role

1. Guessing attacks:
    1. including ‘spouse’ attacks
    2. Use of guessable information about the user
2. Shoulder-surfing attacks
3. Skimming attacks (e.g. at ATMs)
4. Sniffing attacks (key loggers, click-and-clack attacks)
5. Spoofing
6. Password re-use

Spoofing

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

Spoofing prevention



1. Display to user the number of failed log-ins:
    1. 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.
2. **Mutual authentication:**
    1. e.g. in a distributed system, the system could be required to authenticate itself to the user.
3. **Trusted path:** guarantee that the user communicates with the true other party (the guard) and not with a spoofing program:
    1. Authenticated connection, and providing confidentiality (secrecy) and integrity.
    2. On a local OS, the guard is often called the *security kernel* or *reference monitor*.
    3. 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 re-use

1. There has been a proliferation of passwords that we have to recall.
2. People tend to re-use them.
3. People tend to share them across multiple services.
4. Large numbers of people use the same passwords for critical services (e.g. bank accounts) and non-critical stuff on the web.

Choosing passwords

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

The generalised Anderson formula

1. Let:
    1. *P* be the **probability that attacker guesses a password** in a specified time period
    2. *G* be the number of guesses that can be tested in one time unit.
    3. *T* be the number of time units during which guessing occurs
    4. *N* be the number of all possible passwords
2. Then, $P \geq TG/N$.
3. This gives a **lower** bound for *P* (a sort of probabilistic best case for the defender) assuming an attacker who just guesses at random.
4. We'd really like an upper bound too.

An example

1. **Set-up**:
    1. Password chars from alphabet with 96 characters ($A = 96$)
    2. No other constraint on strings except length; our system will allow all passwords no greater than a certain length S.
    3. 10^4 guesses can be tested per second.
    4. Security engineer wants a probability of no greater than a half that there is a successful guess in 365 days.
2. **Problem synopsis**: Find *necessary* condition on S to give $0.5 \geq P$.
3. By Anderson's formula ($P \geq TG/N$), we must have:

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

4. There are $N = \sum_{i=0}^{S} A^i$ passwords of length less than or equal to S.
    1. Because there are $A^S = A \times A \times \ldots \times A$ (with S copies of A) passwords of length S.
    2. So, we need S such that $\sum_{i=0}^{S} 96^i \geq 6.31 \times 10^{11}$.
    3. Calculation shows that we need $S \geq 6$.

Entropy

1. Entropy is a concept from information theory.
2. It is about the information that can be inferred from some data using its probabilistic structure .
3. People speak about the **entropy** of passwords:
    1. This should be a quantity that describes the probability distribution of passwords over the password space.
    2. They speak about `*n bits of entropy/strength*', by which they mean a uniform distribution over passwords that have length *n*.
    3. So, there are 2^n passwords and the probability of guessing a password at random is $1/2^n$ with one guess.

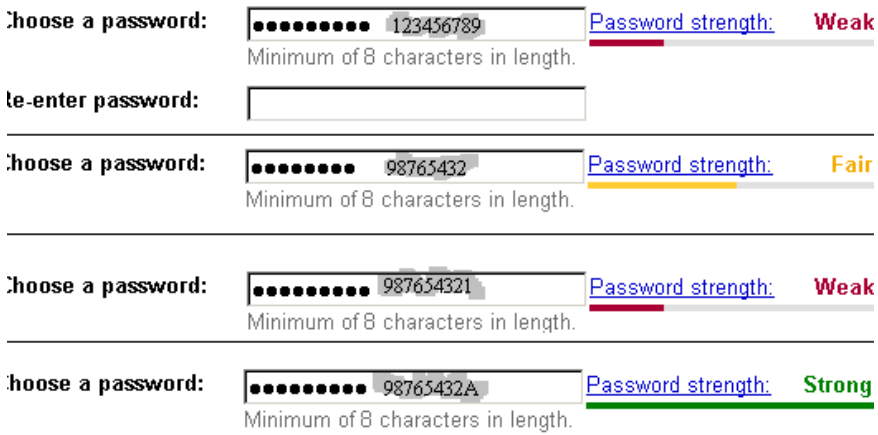
Maximum entropy

1. Each password *a* in the password space *A* gets chosen with probability *P(a)*.
2. Let the expected (average) time required to guess a password be *T*.
    1. Note that this depends upon *P*.
    2. That is, on how probability of each passwords is spread across the space.
3. **Theorem**: *T* is a maximum when *P* is uniform.
    1. *P* is uniform when all passwords are equiprobable.
    2. *P(a) = 1/N,* for all *a*.

Problems

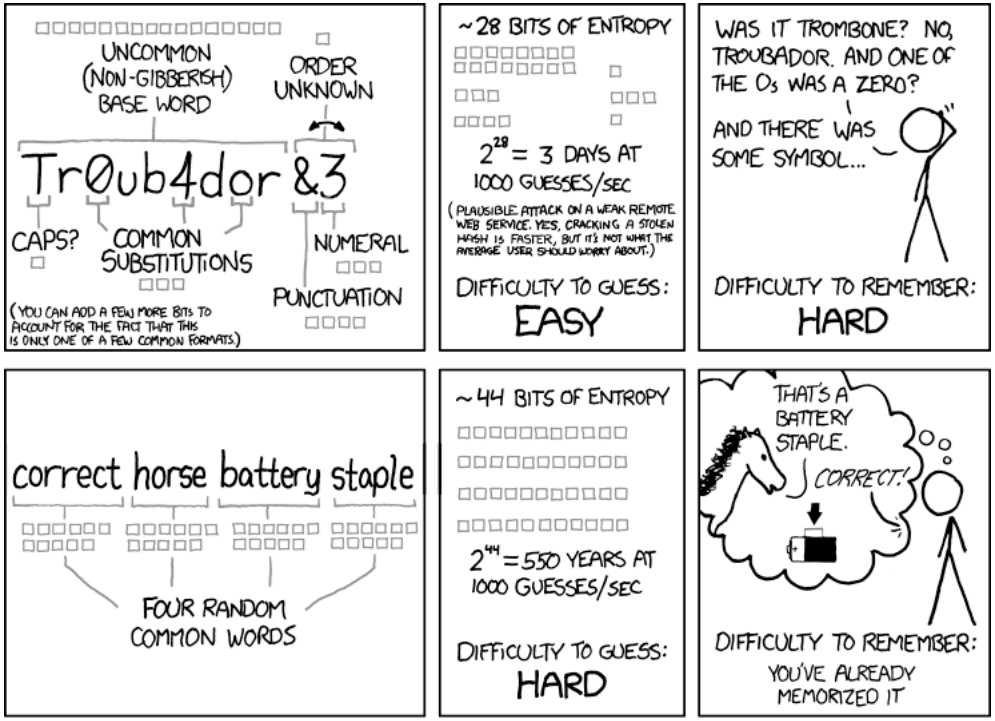
1. The above tells us how ideal beings would select their passwords.
2. Humans don't choose passwords at random:
    1. And find it difficult to do so for long passwords.
    2. Choose pronounceable passwords (phonemes)
    3. Use names, words, user names, keyboard patterns, variants of old/other passwords.
3. A nice (non-examinable) visual demonstration is in the heat maps at: https://blog.qualys.com/securitylabs/2012/07/12/discovered-patterns-in-numeric-passwords-raise-new-questions
4. Want passwords that are easy to remember (usability), but hard to guess (absolute security).

Password strength checks



1. **Proactive password checkers** block users from selecting poor passwords.
2. Can also provide users creating new passwords with easy-to-understand feedback regarding candidate passwords.
    1. Some systems only provide advice to the user.
3. The example above appears to be a mix (blocking and advice).

Password strength



Managing passwords

1. Passwords meant to be secret.
2. Just the right passwords needs to be in just the right places at just the right times (and no-where else).
3. First problem: **bootstrap**.
    1. Get the passwords to the right places
    2. E.g. Users have to come to office to collect
    3. Send by mail, email, phone
    4. Remote user sets password themselves, on a web page?
        1. How do they authenticate before doing that?
        2. See the Wired guy hack later.
    5. Who might collect? Who might intercept?

Managing passwords

1. Possible bootstrap solutions:
    1. Phone: get user to call you from a trusted telephone number. Then call them back there.
    2. Call someone else who can authenticate user, e.g., their manager.
    3. Single-use set-up password:
        1. I have to change immediately after first successful login.
    4. Mail: use personal (secure) delivery.
    5. Different channel confirmation, e.g., password on web-page, followed by confirmation by SMS.

Credential recovery

1. You forget your password. You need a new one. There will be a procedure for getting it back to you or getting you new credentials.
2. Sometimes this is partly-automated using back-up authentication and complementary information.
    1. Even back-up info may be lost or forgotten.
    2. Ultimately, need a full process for credential recovery.
    3. Similar methods to bootstrapping.
3. Many organizations have a unit to do password support.
    1. Personnel require vetting
    2. Personnel require training
    3. Time consuming for the organization
    4. Can be very expensive.

Credential recovery (cont'd)

4. **Important point for security management**:
    1. legitimate users may be unable to gain access.
    2. The security solution must be able to handle this effectively, in particular in a way that scales across the demand (e.g. number of users).

Single Sign-on (SSO)

1. We use many IT systems with the same owner. Example, for some corporate user:
    1. Enter password 1 for workstation
    2. Enter password 2 for network access
    3. Enter password 3 for server access
    4. Enter password 4 for access to database management system
    5. Enter password 5 to open a table in the database.
2. Tedious to keep authenticating for every IT system separately.
3. Various problems arise from use of multiple passwords.
4. So just get the user to authenticate once, and then pass-on this fact appropriately.

Single Sign-on trade off

1. Introduces vulnerabilities.
    1. Possibly leads to less fine-grained control than separate systems if poorly implemented.
        1. Need to make sure access controls for separate systems are not allowing access to system *B* to those whose sign-on should only allow them into *A*.
        2. Okay provided we have not committed the sin of confusing authentication and access control.
    2. Protection problems
        1. Storage, caching, use, etc. by one part of system must not be weaker than requirements of another part of system.
    3. The password becomes a juicier target (more power).
2. Another trade-off. Gollmann calls this an instance of the **curse of convenience.**

Biometrics

1. These belong to the `something you are' factor.
2. E.g. fingerprints, voice recognition, iris scans, etc.
3. Helps to solve many of the usability problems with passwords.
4. But there are problems associated with using biometrics for authentication.

Biometric problems

1. Problems with **false positives** and **false negatives**.
2. Problems with users who have attributes that the system can't recognize – **failures to enroll.**
3. Problem that the complementary data that must be stored can't just be hashed. We must be able to see if authentication data supplied is close to the stored data (not an exact match).
    1. There may be some advanced way around this using the very new homomorphic encryption methods.
4. Therefore, we *must store sensitive private data*.
5. Could be difficult to revoke credentials and issue new ones.
    1. Relies on using only partial biometric data, e.g. specified parts of fingerprint that can later be changed.