Lecture 17, May 8, 2026

Pronounceable Passwords

- Generate phonemes randomly
  - Phoneme is unit of sound, eg. *cv*, *vc*, *cvc*, *vcv*
  - Examples: helgoret, juttelon are; przbqxdfl, zxrptglfn are not
- Problem: too few
- Solution: key crunching
  - Run long key through hash function and convert to printable sequence
  - Use this sequence as password

Pronounceable Passwords

- Bigger problem: distribution of passwords
  - Probabilities of selection of particular phonemes, hence passwords, not equiprobable
  - Generated passwords tend to cluster; if an attacker finds a cluster with passwords user is likely to select, this reduces search space greatly

User Selection

- Problem: people pick easy to guess passwords
  - Based on account names, user names, computer names, place names
  - Dictionary words (also reversed, odd capitalizations, control characters, “elite-speak”, conjugations or declensions, swear words, Torah/Bible/Koran/… words)
  - Too short, digits only, letters only
  - License plates, acronyms, social security numbers
  - Personal characteristics or foibles (pet names, nicknames, job characteristics, *etc*.

Picking Good Passwords

- “WtBvStHbChCsLm?TbWtF.+FSK”
  - Intermingling of letters from Star Spangled Banner , some punctuation, and author’s initials
- What’s good somewhere may be bad somewhere else
  - “DCHNH,DMC/MHmh” bad at Dartmouth (“Dartmouth College Hanover NH, Dartmouth Medical Center/Mary Hitchcock memorial hospital”), ok elsewhere (probably)
- Why are these now bad passwords? ☹

Proactive Password Checking

- Analyze proposed password for “goodness”
  - Always invoked
  - Can detect, reject bad passwords for an appropriate definition of “bad”
  - Discriminate on per-user, per-site basis
  - Needs to do pattern matching on words
  - Needs to execute subprograms and use results
    - Spell checker, for example
  - Easy to set up and integrate into password selection system

Example: OPUS

- Goal: check passwords against large dictionaries quickly
  - Run each word of dictionary through k different hash functions h_1, …, h_k producing values less than n
  - Set bits h_1, …, h_k in OPUS dictionary
  - To check new proposed word, generate bit vector and see if *all* corresponding bits set
    - If so, word is in one of the dictionaries to some degree of probability
    - If not, it is not in the dictionaries

Example: *passwd+*

- Provides little language to describe proactive checking
  - `test length("$p") < 6`
    - If password under 6 characters, reject it
  - `test infile("/usr/dict/words", "$p")`
    - If password in file /usr/dict/words, reject it
  - `test !inprog("spell", "$p", "$p")`
    - If password not in the output from program spell, given the password as input, reject it (because it's a properly spelled word)

Passphrases

- A password composed of multiple words and, possibly, other characters
- Examples:
  - “home country terror flight gloom grave”
    - From Star Spangled Banner, third verse, third and sixth line
  - “correct horse battery staple”
    - From *xkcd* (#936)
- Caution: the above are no longer good passphrases

Remembering Passphrases

- Memorability is good example of how environment affects security
  - Study of web browsing shows average user has 6-7 passwords, sharing each among about 4 sites (from people who opted into a study of web passwords)
    - Researchers used an add-on to a browser that recorded information about the web passwords but *not* the password itself
- Users tend not to change password until they know it has been compromised
  - And when they do, the new passwords tend to be as short as allowed
- Passphrases seem as easy to remember as passwords
  - More susceptible to typographical errors
  - If passphrases are text as found in normal documents, error rate drops

Password Manager (Wallet)

- A mechanism that encrypts a set of user's passwords
- User need only remember the encryption key
  - Sometimes called "master password"
  - Enter it, and then you can access all other passwords
- Many password managers integrated with browsers, cell phone apps
  - So you enter the master password, and password manager displays the appropriate password entry
  - When it does so, it shows what the password logs you into, such as the institution with the server, and hides the password; you can then have it enter the password for you

Dictionary Attacks

- Trial-and-error from a list of potential passwords
  - *Off-line*: know *f* and *c*'s, and repeatedly try different guesses $g \in A$ until the list is done or passwords guessed
    - Examples: *crack*, *john-the-ripper*
  - *On-line*: have access to functions in *L* and try guesses *g* until some $l(g, c)$ succeeds
    - Examples: trying to log in by guessing a password

Salting

- Goal: slow dictionary attacks
- Method: perturb hash function so that:
  - Parameter controls *which* hash function is used
  - Parameter differs for each password
- Method: perturb the password
  - Add extra characters or bits to the data before hashing
- Either way, given *n* password hashes, and therefore *n* salts, need to hash each guess *n* times

Examples

- Vanilla UNIX method
  - Use DES to encipher 0 message with password as key; iterate 25 times
  - Perturb E table in DES in one of 4096 ways
    - 12 bit salt flips entries 1–11 with entries 25–36
- Linux/*BSD methods
  - Prepend or append salt to password, then hash the result

Using Time

Anderson's formula:

- P probability of guessing a password in specified period of time
- G number of guesses tested in 1 time unit
- T number of time units
- N number of possible passwords ($|A|$)
- Then $P \geq TG/N$

Example

- Goal
  - Passwords drawn from a 96-char alphabet
  - Can test 10^{15} guesses per second
  - Probability of a success to be 0.5 over a 365 day period
  - What is minimum password length?
- Solution
  - $N \geq TG/P = (365 \times 24 \times 60 \times 60) \times 10^{15}/0.5 = 6.31 \times 10^{23}$
  - Choose s such that $\sum_{j=0}^{s} 96^j \geq N$
  - So $s \geq 13$, meaning passwords must be at least 3 chars long

Guessing Through *L*

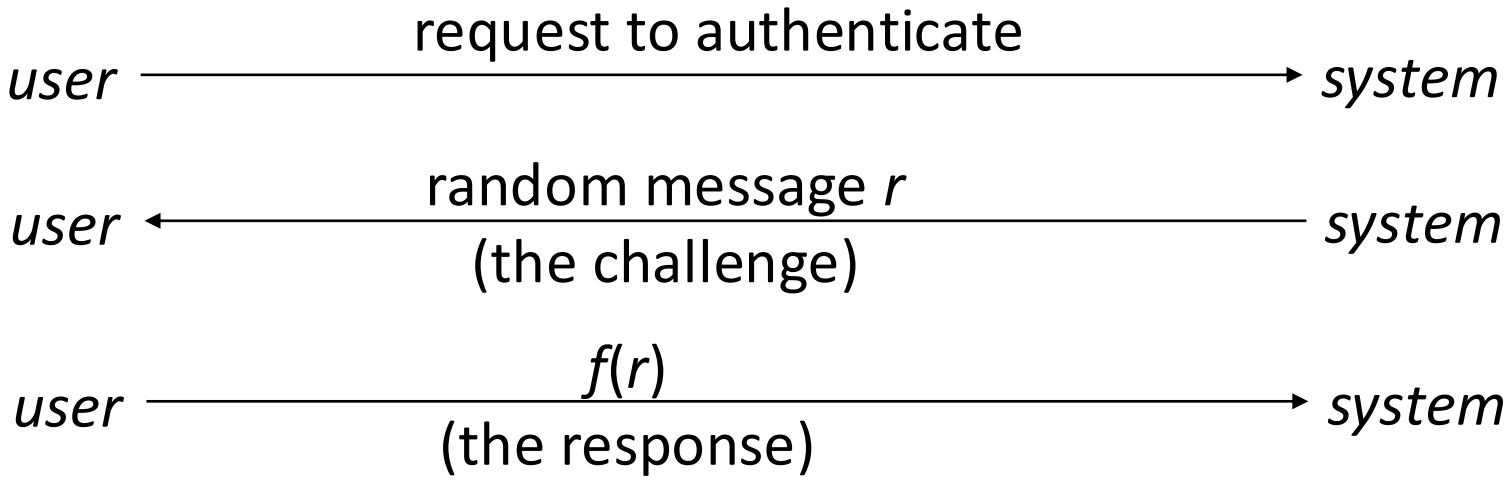
- Cannot prevent these
  - Otherwise, legitimate users cannot log in
- Make them slow
  - Backoff
  - Disconnection
  - Disabling — be very careful with administrative accounts!
  - Jailing — allow in, but restrict activities

Password Aging

- Force users to change passwords after some time has expired
- How do you prevent re-used passwords?
  - Record previous passwords
  - Block changes for a period of time
- Give users time to think of good passwords
  - Don’t force them to change before they can log in
  - Warn them of expiration days in advance

Challenge-Response

- User, system share a secret function *f* (in practice, *f* is a known function with unknown parameters, such as a cryptographic key)



Pass Algorithms

- Challenge-response with the function *f* itself a secret
- Example:
  - Challenge is a random string of characters such as “abcdefg”, “ageksido”
  - Response is some function of that string such as “bdf”, “gkip”
- Usually used in conjunction with fixed, reusable password

One-Time Passwords

- Password that can be used exactly *once*
  - After use, it is immediately invalidated
- Challenge-response mechanism
  - Challenge is number of authentications; response is password for that particular number
- Problems
  - Synchronization of user, system
  - Generation of good random passwords
  - Password distribution problem

S/Key

- One-time password scheme based on idea of Lamport
- *h* is a cryptographic hash function (SHA-256, for example)
- User chooses initial seed *k*
- System calculates:

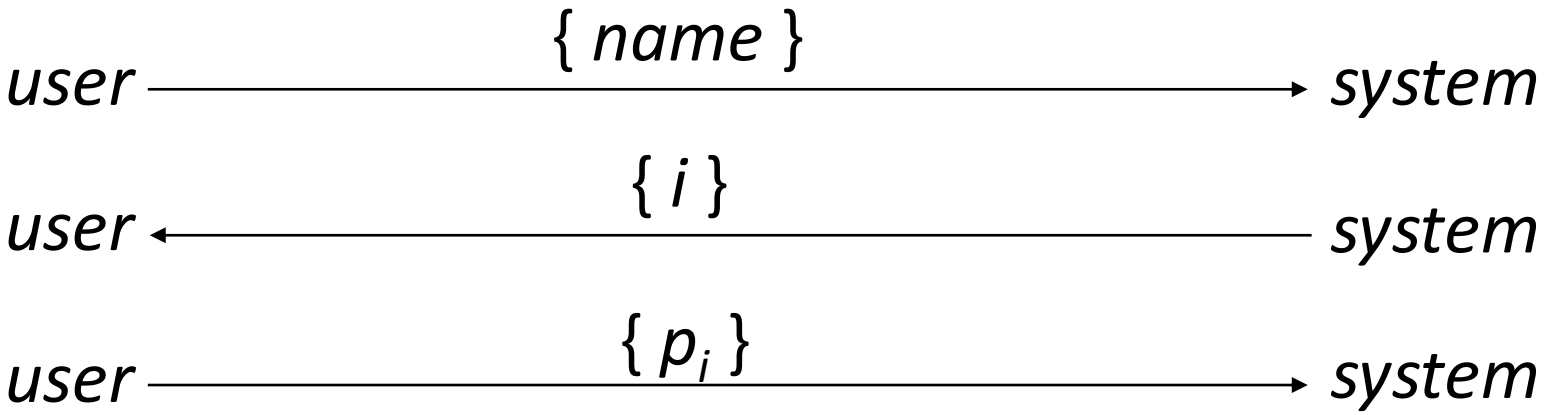
$$h(k) = k_1,\ h(k_1) = k_2,\ \ldots,\ h(k_{n-1}) = k_n$$

- Passwords are these in reverse order:

$$p_1 = k_n,\ p_2 = k_{n-1},\ \ldots,\ p_{n-1} = k_2,\ p_n = k_1$$

S/Key Protocol

System stores maximum number of authentications n, number of next authentication i, last correctly supplied password p_{i-1}.



System computes $h(p_i)$; if correct, you get $h(p_i) = h(k_{n-i+1}) = k_{n-i} = p_{i-1}$.
This matches what is stored, system replaces p_{i-1} with p_i and increments i.
If not correct, the computed hash does not match what is stored.

Hardware Support

- Token-based
  - Used to compute response to challenge
    - May encipher or hash challenge
    - May require PIN from user
- Temporally-based
  - Every minute (or so) different number shown
    - Computer knows what number to expect when
  - User enters number and fixed password
- Example: Duo uses temporally-based number
  - May not show; a tap will send it

Challenge Response and Dictionary Attacks

- Same as for fixed passwords
  - Attacker knows challenge *r* and response *f*(*r*); if *f* encryption function, can try different keys
  - May only need to know *form* of response; attacker can tell if guess correct by looking to see if deciphered object is of right form
  - Example: Kerberos Version 4 used DES, but keys had 20 bits of randomness; Purdue attackers guessed keys quickly because deciphered tickets had a fixed set of bits in some locations

Biometrics

- Automated measurement of biological, behavioral features that identify a person
  - Fingerprints: optical or electrical techniques
    - Maps fingerprint into a graph, then compares with database
    - Measurements imprecise, so approximate matching algorithms used
  - Voices: speaker verification or recognition
    - Verification: uses statistical techniques to test hypothesis that speaker is who is claimed (speaker dependent)
    - Recognition: checks content of answers (speaker independent)

Other Characteristics

- Eyes: patterns in irises unique
  - Measure patterns, determine if differences are random; or correlate images using statistical tests
- Faces: image, or specific characteristics like distance from nose to chin
  - Lighting, view of face, other noise can hinder this
- Keystroke dynamics: believed to be unique
  - Keystroke intervals, pressure, duration of stroke, where key is struck
  - Statistical tests used

Cautions

- These can be fooled!
  - Assumes biometric device accurate *in the environment it is being used in!*
  - Assumes biometric is consistent enough to give accurate readings
  - Transmission of data to validator is tamperproof, correct

Location

- If you know where user is, validate identity by seeing if person is where the user is
  - Requires special-purpose hardware to locate user
  - Example: GPS (global positioning system) device gives location signature of entity
    - Host uses LSS (location signature sensor) to get signature for entity

Multiple Methods

- Example: “where you are” also requires entity to have LSS and GPS, so also “what you have”
- Can assign different methods to different tasks
  - As users perform more and more sensitive tasks, must authenticate in more and more ways (presumably, more stringently) File describes authentication required
    - Also includes controls on access (time of day, *etc*.), resources, and requests to change passwords
- Example: Pluggable Authentication Modules (PAM)

PAM: Used in Linux, *BSD

- Idea: when program needs to authenticate, it checks central repository for methods to use
- Library call: *pam_authenticate*
  - Accesses file with name of program in */etc/pam_d*
- Modules do authentication checking
  - *sufficient*: succeed if module succeeds
  - *required*: fail if module fails, but all required modules executed before reporting failure
  - *requisite*: like *required*, but don't check all modules
  - *optional*: invoke only if all previous modules fail

Example PAM File

```
auth  sufficient /usr/lib/pam_ftp.so
auth  required   /usr/lib/pam_unix_auth.so use_first_pass
auth  required   /usr/lib/pam_listfile.so onerr=succeed \
          item=user sense=deny file=/etc/ftpusers
```

For ftp:

1. If user “anonymous”, return okay; if not, set PAM_AUTHTOK to password, PAM_RUSER to name, and fail
2. Now check that password in PAM_AUTHTOK belongs to that of user in PAM_RUSER; if not, fail
3. Now see if user in PAM_RUSER named in /etc/ftpusers; if so, fail; if error or not found, succeed

Access Control Lists

- Columns of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

ACLs:

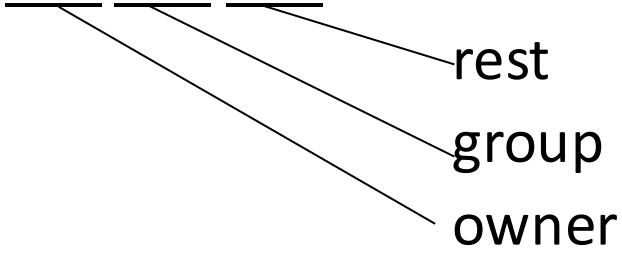
- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

Default Permissions

- Normal: if not named, *no* rights over file
  - Principle of Fail-Safe Defaults
- If many subjects, may use groups or wildcards in ACL
  - UNICOS: entries are (*user*, *group*, *rights*)
    - If *user* is in *group*, has rights over file
    - '*' is wildcard for *user*, *group*
      - (holly, *, r): holly can read file regardless of her group
      - (*, gleep, w): anyone in group gleep can write file

Abbreviations

- ACLs can be long … so combine users
  - UNIX: 3 classes of users: owner, group, rest
  - rwx rwx rwx



  - Ownership assigned based on creating process
    - Most UNIX-like systems: if directory has setgid permission, file group owned by group of directory (Solaris, Linux)

ACLs + Abbreviations

- Augment abbreviated lists with ACLs
  - Intent is to shorten ACL
- ACLs override abbreviations
  - Exact method varies
- Example: Extended permissions (Linux, FreeBSD, others)
  - Minimal ACLs are abbreviations, extended ACLs give specific users, groups permissions
  - Extended ACL entries give rights provided those rights are in mask

Minimal and Extended ACL

user *heidi*, group *family* owns file with permissions:

```
user::rw-
user:skyler:rwx
group::rw-
group:child:r--
mask::rw-
other::r--
```

- *heidi* can read, write file (first line)
- *matt*, not in group *child*, can read file (last line)
- *skyler* can read, write file (second line masked by fifth line)
- *sage*, in group *family*, can read, write the file (third line masked by fifth line)
- *steven*, in group *child*, can read file (fourth line masked by fifth line)

ACL Modification

- Who can do this?
  - Creator is given *own* right that allows this
  - System R provides a *grant* modifier (like a copy flag) allowing a right to be transferred, so ownership not needed
    - Transferring right to another modifies ACL

Privileged Users

- Do ACLs apply to privileged users (*root*)?
  - Solaris: abbreviated lists do not, but full-blown ACL entries do
  - Other vendors: varies

Groups and Wildcards

- Classic form: no; in practice, usually
- UNICOS:
  - `holly : gleep : r`
    user *holly* in group *gleep* can read file
  - `holly : * : r`
    user *holly* in any group can read file
  - `* : gleep : r`
    any user in group *gleep* can read file

Conflicts

- Deny access if any entry would deny access
  - AIX: if any entry denies access, *regardless or rights given so far*, access is denied
- Apply first entry matching subject
  - Cisco routers: run packet through access control rules (ACL entries) in order; on a match, stop, and forward the packet; if no matches, deny
    - Note default is deny so honors principle of fail-safe defaults

Handling Default Permissions

- Apply ACL entry, and if none use defaults
  - Cisco router: apply matching access control rule, if any; otherwise, use default rule (deny)
- Augment defaults with those in the appropriate ACL entry
  - AIX: extended permissions augment base permissions

Revocation Question

- How do you remove subject's rights to a file?
  - Owner deletes subject's entries from ACL, or rights from subject's entry in ACL
- What if ownership not involved?
  - Depends on system
  - System R: restore protection state to what it was before right was given
    - May mean deleting descendent rights too …

Windows 10 NTFS ACLs

- Different sets of rights
  - Basic: read, write, execute, delete, change permission, take ownership
  - Generic: no access, read (read/execute), change (read/write/execute/delete), full control (all), special access (assign any of the basics)
  - Directory: no access, read (read/execute files in directory), list, add, add and read, change (create, add, read, execute, write files; delete subdirectories), full control, special access

Accessing Files

- User not in file's ACL nor in any group named in file's ACL: deny access
- ACL entry denies user access: deny access
- Take union of rights of all ACL entries giving user access: user has this set of rights over file

Example

- Paul, Quentin in group *students*
- Quentin, Regina in group *staff*
- ACL entries for *E:\stuff*
  1. *staff,* create files/write data, allow
  2. Quentin, delete subfolders and files, allow
  3. *students,* delete subfolders and files, deny
- Regina can create files or subfolders (1)
- Quentin cannot delete subfolders and files
  - Even with 2; Quentin in *students*, and explicit deny in 3 overrides allow in 2

Example (*con't*)

- Regina wants to create folder *E:\stuff\plugh* and set it up so:
  - Paul doesn't have delete subfolders and files access
  - Quentin has delete subfolders and files access
- How does she do this?

How She Does It

Inherited from *E:\stuff*:

staff, create files/write data, allow

Quentin, delete subfolder and files, allow

~~*students*, delete subfolder and files, deny~~

Paul, delete subfolders and files, deny