

CMPUT 299 (B1) Winter 2008

# **Security in a Networked World**

*Authentication*

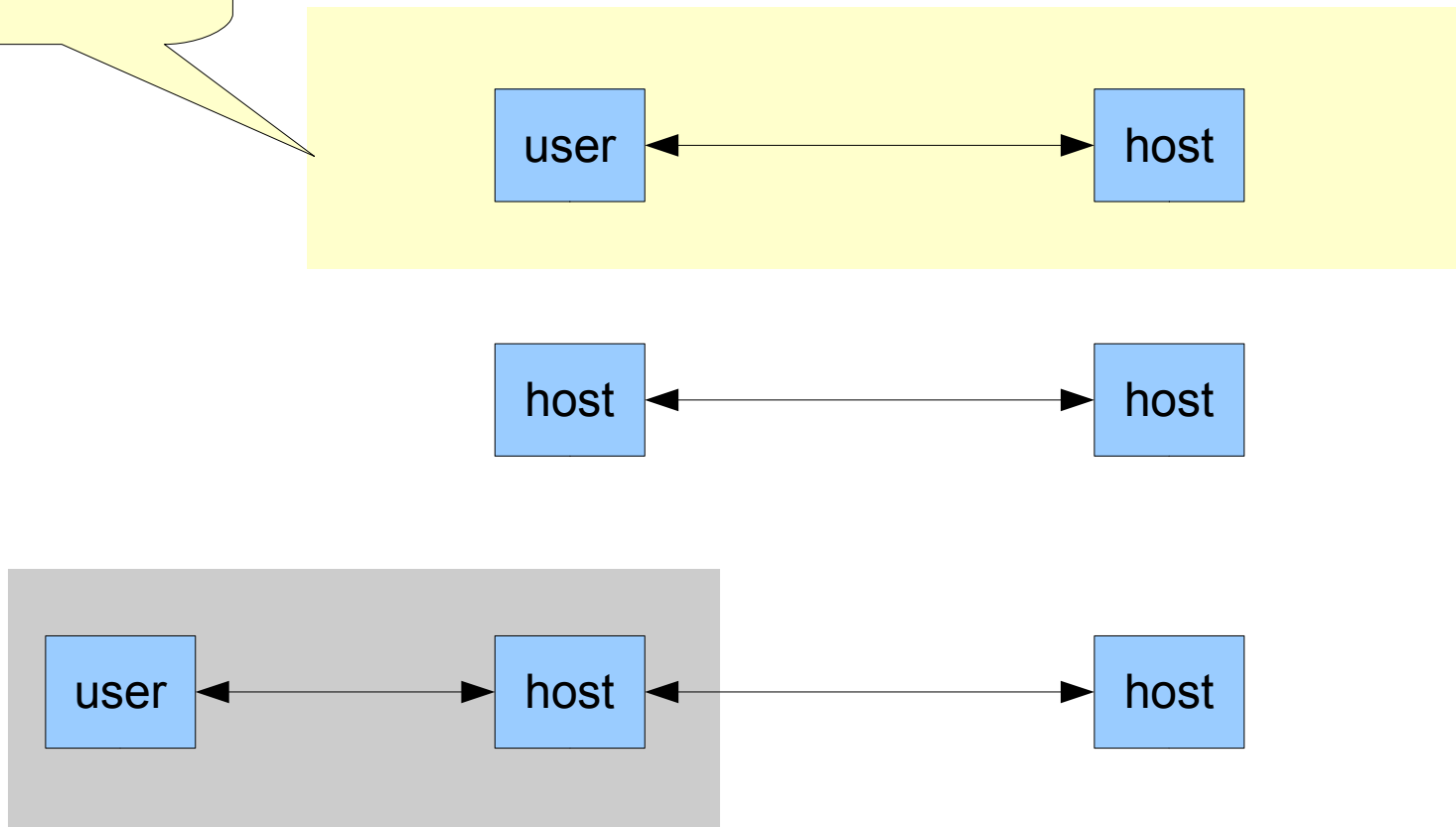
yannis@cs.ualberta.ca

# Password Authentication

- ▶ Authentication is based on something the entity being authenticated possesses (e.g., a security token, a smart card, etc.), or knows (e.g., a password), or is part of its unique identity (e.g., its retina, fingerprint, etc.).
- ▶ Passwords have been used since ancient times.
- ▶ Passwords are susceptible to *eavesdropping* and *replay/reuse*. (Note: a password may or may not be a *key*.)
- ▶ Solution: change the passwords to restrict exposure. (Roman army units would change it daily.)
- ▶ Even better solution: use a password only once.

# Man & Machine Authentication

The most problematic.



# The Limitations of Passwords

- ▶ Use by humans means:
  - ▶ Limited in length and poorly chosen.
- ▶ Usual password cracking approach:
  - ▶ Use dictionaries (in many languages).
  - ▶ Use versions of proper words (e.g. leetspeak).
  - ▶ Use user context (work, family, etc.) words.
  - ▶ Word-like constructs (pronounceable random words).
- ▶ How good are they:
  - ▶ Morris & Thompson in 1979 reported 86% success!
  - ▶ In recent years, commercial products such as the Password Recovery Toolkit (from [www.accessdata.com](http://www.accessdata.com)) has been reported to have a >50% success rate.

# On-line vs. Off-line Password Cracking

On-line:

- ▶ Systems subjected to attacks can at least log attempts.
- ▶ Possibly deny access after a number of failed attempts.
  - ▶ There is a catch: DoS
- ▶ Slow down attempts (not noticeable to normal users).
  - ▶ Slowed down crypto implementation (purposefully).
  - ▶ CAPTCHAs (but requires proper UI).
  - ▶ Delayed rejection of invalid username until password is entered as well (do not tell which is wrong).
  - ▶ Moreover, one should be unable to infer anything from the timing between username and password prompt.

## On-line vs. Off-line Password Cracking (cont'd)

- ▶ There is no limit on how many resources can be allocated to crack passwords off-line. [But one has to obtain the password hashes via another vulnerability.]
- ▶ Not restricted to using slow implementations of the cryptographic primitives. Hardware implementation is also possible, e.g. on FPGAs.
- ▶ It is possible to also exploit some weaknesses of the underlying cryptographic algorithm. [We need cryptography because the password should not be stored in plaintext.]

# The DES-based Unix Password Hash

- ▶ Converting a symmetric block cipher algorithm (including DES) to a hash function is easy:
  - ▶ Suppose encryption is denoted by  $E_k(P)$  where  $k$  is the key and  $P$  is the plaintext. There also exists a corresponding decryption  $D_k(C)$  where  $C$  is the ciphertext.
  - ▶ To convert to a hash function: set  $P$  to a *known* string (could be NULL), use the password for  $k$ . So we are essentially defining a hash function  $H(k) = E_k(\text{NULL})$ . What is stored for user  $u$  is  $H(k)$  and *not*  $k$ .
  - ▶ Authenticating a user claiming to be  $u$  given a password  $k'$  is answering: “ $D_{k'}(H(k)) == \text{NULL} ?$ ”

# Salt

- ▶ A dictionary-based attack by computation of hash values for a large dictionary of “words” would allow a simple lookup to determine the key  $k$  given access to  $H(k)$ . (Or one could go through a more space-demanding one-time only brute force pre-computation.)
- ▶ Add salt: combine a short random string to  $k$ . The random string is called the “salt,”  $s$ .
- ▶ What is stored for user  $u$  is  $(s, H_{s||k}(P))$  where  $||$  is an expression to denote the combination of  $s$  and  $k$  (could be appending/pre-pending/other). User  $u$  authenticates with  $k'$  if “ $D_{s||k'}(H(s||k)) == \text{NULL}$ ”

# Salt's Properties

- ▶ Because  $s$  is chosen (pseudo-)randomly, hence:
  - ▶ The same  $k$  on different hosts (hence very likely different  $s$ ) will almost certainly map to different stored hashes.
  - ▶ Assuming  $k$ 's size is  $|k|$  bits, and  $s$ 's size is  $|s|$  then there are  $2^{|s|}$  hashes for the same  $k$ , making a dictionary attack much more difficult.
  - ▶ If we are generating  $s$  randomly (well, at least for each host) why not generate  $k$  also randomly?
  - ▶ Two issues: usability and possibly problematic Random Number Generator (RNG).

## Digression: Randomness

- ▶ A truly random 64 bit integer:
  - ▶ translates to about 20 decimal digits, or
  - ▶ 11 characters from [a-z,A-Z,0-9]+punctuation, or
  - ▶ 16 characters for pronounceable password.
  - ▶ All are difficult for a human to remember.
- ▶ Crucially dependent on (Pseudo)RNG.
  - ▶ Poorly designed PRNGs quickly cycle.
  - ▶ Morris & Thompson report in 1979 that a random password generator for producing 8 character long lower case alpha + numeric values was producing only  $2^{15}$  distinct outputs.
  - ▶ Good PRNGs are notoriously hard to design.

## Digression: Randomness (cont'd)

- ▶ Usual workings of a feedback PRNG (simplified):
  - ▶ An “internal” state  $S$  is maintained.
  - ▶  $S$  is changed in each invocation of the PRNG.
  - ▶ The output  $U$  is a function of the state  $S$ .
  - ▶ The initial  $S$  is derived from the “seed” of the PRNG.
  - ▶ By resetting  $S$  to the initial “seeded” value one can restart the PRNG, i.e., it can repeat the same sequence of (“random”) numbers.
  - ▶ In summary: (init:  $S = f(\text{seed})$ ; it could be  $S = \text{seed}$ ; !)
    - ▶  $(U, S') = \text{PRNG}(S)$  /\*  $U$  is passed to user,  $S'$  is the new  $S$  \*/
    - ▶  $S = S'$

How is the initial seed formed? Are all  $U$ s distinct? Is  $S$  “large” enough (bits)?

## Limitations of Unix DES-based Password Hash

- ▶ The keys in DES are 8 characters long but only 7 of the 8 bits are used from each byte, hence 56 bits long. Longer passwords are truncated to fit the DES key length requirement.
- ▶ A two byte (but only 6 bits of each used) salt is used to perturb the DES stages (specifically bits  $i$  and  $i+24$  in the DES E-Box output are swapped if the  $i$ -th bit in the salt is set). [This was to render hardware DES implementations unusable back when it was proposed. Nowadays irrelevant.]
- ▶ After 25 repetitions of the DES stage, we get 64 bits + 12 (salt) bits = 76 bits  $\rightarrow$  13 characters



# Lessons Unlearned

- ▶ Web application authentication schemes repeat the same mistakes of the past:
  - ▶ Separate messages for identification (username) failure and authentication (password) failure.
    - ▶ Either on rendered page or as URL parameters.
  - ▶ Allow dubious approaches to authentication.
    - ▶ Allow users to create accounts without intervention. The assumption is that user accounts are casual and hence not valuable. However, a new user account allows access to more software functionality, and more possibly exploitable weaknesses.
    - ▶ Philosophical question: What good is the enforcement of strong password selection when literally *anyone* can obtain an account just for the asking?



## Lessons Unlearned (cont'd)

- ▶ Initial trivial passwords are a fact of life for almost all commodity networking and consumer grade equipment:
  - ▶ All firewalls, routers, networked media players etc. have a default password.
    - ▶ <http://www.phenoelit-us.org/dpl/dpl.html> for examples.
    - ▶ Regular consumers are less diligent than sysadmins.
    - ▶ Restrictions to “local subnet” access usually inadequate
      - ▶ Weakly encrypted wireless access point within the local subnet.
  - ▶ Best practice: unique per-unit default password.
    - ▶ It cannot be the serial number. (Predictable)
    - ▶ Affix (long) password along with serial number sticker.

# “Physical Security is the Only/Best Security”

- ▶ (Implicit) authentication based on the fact that there is only one way (path, port, cable, etc.) from which the user could possibly communicate, imposed by the physical design of the system. [In these cases even identity is not required. If it is connected on port X, it must be entity Y.]
- ▶ Today, claims of physical security (unless you see it with your own eyes) should be approached with a grain of salt. You do not know what is happening “at the other end of the wire.” In fact, we can no longer talk about (dedicated) physical wire in almost all settings.



# The Telephone System

- ▶ We trust that in a telephone system, when we call a particular number, it is precisely that number that will be called and there is a physical path established between the endpoints (in the old days: a dedicated circuit). (We rely on the integrity of call routing.)
- ▶ Digital exchanges added flexibility to call routing and replaced physical with logical paths, but the essence of the guarantee remained the same.
- ▶ Receiving a call is a different story. Some systems authenticate using the *Caller ID*, but a Caller ID is not a password (it is an identifier) and is incorrectly relied upon for authentication.



## Pitfalls of Caller ID

- ▶ Some systems (e.g., cellular provider voicemail services) use Caller ID as authentication attribute. They *usually* do not require PIN (or password) if Caller ID indicates a number identical to the called number. It is assumed it is the owner of the voicemail box who calls!
- ▶ Caller ID is easily manipulated (*telespoof*, *camophone*, or via VoIP PBX control software, like *Asterisk*, with suitable equipment).
- ▶ Corollary: numerous cellular voicemail systems are vulnerable to anyone who cares to manipulate his/her Caller ID identifier.

# The Unix r-commands

- ▶ Early networked Unix systems attempted to provide a “single logon” facility by means of r-commands (`rsh`, `rlogin`, `rsh`, ...). Password authentication is not requested if the login request comes from a “trusted” system.
- ▶ Trusted accounts are indicated as *host username* pairs (if login comes from *host* and the remote user is *username*, no password is requested). The trusted accounts are stored in per-user `.rhosts` file and in `/etc/hosts.equiv`
- ▶ The scheme is vulnerable to spoofing attacks. Collections of hosts on a LAN are usually all vulnerable once one is compromised.

# Reducing Programming Effort

- ▶ Authentication is needed in many applications.
- ▶ Developers that are not experts in security usually re-invent the wheel, writing authentication routines from scratch. Occasionally, disregarding the evolution of techniques in this area.
- ▶ The challenge is to write “libraries” that are usable to any program wishing to use authentication services. Reduce burden on application programmer.
- ▶ An equal challenge is that the site owner should have control over how strong the authentication for an application should be.

# Pluggable Authentication Modules (PAMs)

- ▶ An architecture integrating various authentication schemes under a uniform interface.
- ▶ Applications do not directly invoke specific authentication mechanisms, they just indicate that they require authentication. Site admins specify what authentication is to be used for each service/program.
- ▶ Multiple authentication techniques can be combined/composed (“stackability”).
- ▶ PAMs are loaded on-demand, meaning that they require the underlying OS to support dynamically loaded libraries.