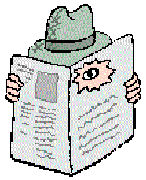


Computer Security

Authentication

What is Authentication?

- Remember, from before:
  - ***Authentication***
    - ***The process of reliably verifying the identity of someone or something***
      - In particular, did a message indeed come from its specified sender?
  - When authenticating messages, we need to guard against:
    - Disclosure of a message to any unauthorized person or system
    - Messages masquerading as being from a source
    - Content modification
      - Was the message changed between sender and recipient?)
    - Source or destination repudiation
      - Can the sender or recipient deny they sent/received the message?

Identification, Authentication, Authorization

- Be sure to distinguish these three concepts:
  - ***Identification***
    - Associating an identity with a subject
  - ***Authentication***
    - Establishing the validity of something, such as an identity
  - ***Authorization***
    - Associating rights or capabilities with a subject

What are we Authenticating?

- ***Authentication of a person***
  - Others know you by your appearance or voice
  - By your picture on an identification badge
  - Other information you have or know
- ***Authentication of a computer***
  - Computer authenticating another computer
    - Print spooler authenticating a printer, etc.
  - Person using a public workstation
    - Workstation will (should) not store authentication information for every user.
    - Person needs to remember the authentication information

Authentication of a Person

- Authentication of a person (a.k.a. human, user) can be achieved through:
  - ***What you know***
    - Passwords; credit card information; SSN, etc.
  - ***What you have***
    - Car, house, or office keys; ATM card
  - ***What you are***
    - Facial or voice recognition (subject to daily change)
    - Written signature (not easily analyzed by computer)
    - DNA
    - Biometric devices
      - Retinal scanner, iris scanner
      - Fingerprints, handprints
      - Keystroke timing

Passwords

- Challenge/Response:
  - "Who goes there -- friend or foe?"
  - "Friend!"
  - "Identify yourself!"
  - "I'm Bryan Higgs"
  - "What's the password of the day?"
  - "Cryptography rocks!"
- If I get the password correct, I am granted access
- If I don't, the consequences may be dire:
  - In a civilian situation, I may be arrested
  - In a military situation, I may be shot!

Passwords

- The problems with passwords (a.k.a. pass phrases) are:
  - Someone may be listening (eavesdropping -- Eve)
  - You may forget the password (especially if it's long and/or cryptic)
  - You may choose a password that's easy to guess
    - Passwords that are hard to guess are often also hard to remember

"Social Engineering"

- From The Jargon Dictionary (http://info.astrian.net/jargon/):
  - **social engineering** *n.* Term used among crackers and samurai for cracking techniques that rely on weaknesses in wetware rather than software; the aim is to trick people into revealing passwords or other information that compromises a target system's security. Classic scams include phoning up a mark who has the required information and posing as a field service tech or a fellow employee with an urgent access problem. See also the tiger team story in the patch entry.
- Adaptation of classic cons to the computer world.

Avoid "Social Engineering" Scams

- Don't give out user names and passwords
  - Administrators do not need them, and should never ask for them.
- Treat incoming phone calls with skepticism!
  - Someone calls you saying they are from the phone company and are testing the line. To complete the test, please dial 90# and hang up. The person calling can now make a phone call that is charged to your phone bill.
  - Someone calls you (or the help desk) saying they are working to fix a server, and ask for a user name and password to use for testing. The person now uses this information to hack the server.
  - These people can be amazing devious, and amazingly plausible!

Avoid "Social Engineering" Scams

- Treat incoming emails with suspicion!
  - My ISP and email providers often send around email telling us that they will never ask us for our password, and warning us of actual, active, scams which appear to come from that ISP/email provider, but in fact don't.
  - I have received countless SPAM email messages purporting to come from Microsoft, saying that the attachment is the latest security update to be applied to my Windows system.
    - Really, it's something that will actually ***undermine*** my system security
    - The graphics in the email were exceedingly well done; they looked so much like the real email messages from Microsoft (complete with logos, look-and-feel, etc.) that I could easily see how they could fool a lot of people.
    - Here's what it looked like...

Avoid "Social Engineering" Scams

From - "Microsoft Customer Support" <xjcwcbgxtftt-fhkrwc@technet.ms.com>
Subject - "Last Security Upgrade" **or** "New Network Update" **or** "net patch"
Attachment - "patch96.zl9" (DO NOT OPEN)
Message type - Text/ HTML
Message body -

Microsoft — All products | Support | Search | Microsoft.com guide
Microsoft home

Microsoft Customer

this is the latest version of security update, the "September 2003, Cumulative Patch" update which fixes all known security vulnerabilities affecting MS Internet Explorer, MS Outlook and MS Outlook Express as well as three new vulnerabilities. Install now to maintain the security of your computer from these vulnerabilities. This update includes the functionality of all previously released patches.

System requirements	Windows 95/98/Me/2000/NT/XP
This upgrade applies to	MS Internet Explorer, version 4.01 and later MS Outlook, version 8.00 and later MS Outlook Express, version 4.01 and later
Recommendation	Customers should install the patch at the earliest opportunity.
How to install	Run attached file. Choose Yes on displayed dialog box.
How to use	You don't need to do anything after installing this item.

Microsoft Product Support Services and Knowledge Base articles can be found on the Microsoft Technical Support web site. For security-related information about Microsoft products, please visit the Microsoft Security Advisor web site, or Contact Us.

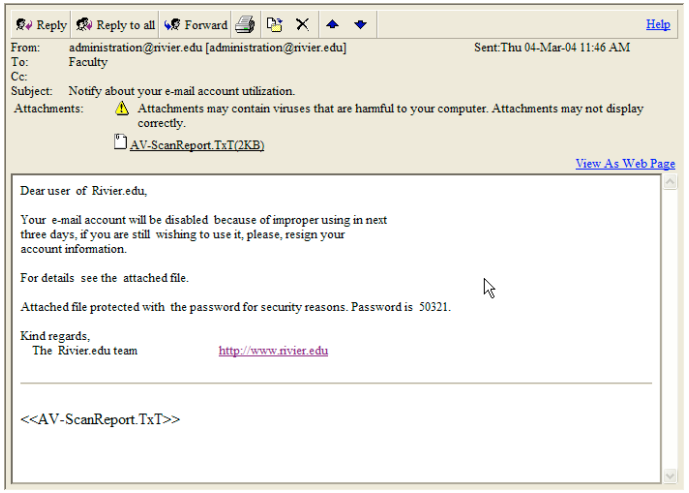
Thank you for using Microsoft products.

Please do not reply to this message. It was sent from an unmonitored e-mail address and we are unable to respond to any replies.

The names of the actual companies and products mentioned herein are the trademarks of their respective owners.

Contact us | Legal | Truste
©2003 Microsoft Corporation. All rights reserved. Terms of Use | Privacy Statement | Accessibility

Other Scams and Tricks



"Dumpster Diving"

dumpster diving */dump'-ster di:'-ving/ n.* **1.** The practice of sifting refuse from an office or technical installation to extract confidential data, especially security-compromising information ('dumpster' is an Americanism for what is elsewhere called a 'skip'). Back in AT&T's monopoly days, before paper shredders became common office equipment, phone phreaks (see phreaking) used to organize regular dumpster runs against phone company plants and offices. Discarded and damaged copies of AT&T internal manuals taught them much. The technique is still rumored to be a favorite of crackers operating against careless targets. **2.** The practice of raiding the dumpsters behind buildings where producers and/or consumers of high-tech equipment are located, with the expectation (usually justified) of finding discarded but still-valuable equipment to be nursed back to health in some hacker's den. Experienced dumpster-divers not infrequently accumulate basements full of moldering (but still potentially useful) cruft.

The Value of Owning a Shredder[1]

- ***To avoid the hazards of dumpster diving, shred important documents before throwing them out!***
  - The notorious hacker, Kevin Mitnick tells us that he was highly successful using dumpster-diving techniques:
    - http://www.nwfusion.com/news/2000/0928mitnick.html
    - http://www.pcplus.co.uk/media/pcplus/pdf/198/198.interview.kevin%20mitnick.pdf
    - http://netsecurity.about.com/cs/bookreviews/gr/aapr032103.htm

[1]Credit must go to Ian Manchester for the title and idea of this slide.

"Shoulder Surfing"

- Someone watches over your shoulder while you type in your user name and password. They later log in and do bad things in your name.
- To stop shoulder surfing:
  - Be aware of your surroundings.
  - If someone is standing too close, ask them to back up.

"Office Surveillance"

- Looking for passwords or clues to passwords in a person's office.
- Stopping Office Surveillance:
  - Memorize your passwords
  - Don't put sticky notes on monitors with the password.
  - Don't put passwords on notes in drawers.
  - Don't write pins on bank-cards
  - Use passwords that are not made from personal data.

Other Things to Avoid

- Don't:
  - Put plaintext passwords in files on your system, or in scripts that you or others might execute.
  - If you've logged into a web site using a browser on a public computer, be sure to clear the history and other context such as cookies that might have been saved on the system.
  - In short, think about what you're doing with your passwords!

Secure Shell (SSH)

- Many utilities on UNIX (and perhaps other) systems (rlogin, telnet, rcp, ftp, etc.) send passwords in plaintext over the network.
- Users of telnet, rlogin, ftp, and other such programs might not realize that their password is transmitted across the Internet unencrypted, but it is.
- A solution to this is the use of Secure Shell (SSH).
- One implementation is OpenSSH:

  "OpenSSH encrypts all traffic (including passwords) to effectively eliminate eavesdropping, connection hijacking, and other network-level attacks. Additionally, OpenSSH provides a myriad of secure tunneling capabilities, as well as a variety of authentication methods."

Password Guessing

- There are two types of ***Password Guessing Attacks***:
  - On-line
    - Attempts to log into the system directly
    - Easy to thwart:
      - Limit number of tries before disabling
        - » Most operating systems implement this
        - » ATM machines eat your card if you type three incorrect PINs
      - Make the system be slow at checking a password to minimize guesses/minute
  - Off-line
    - If attacker has some information that is derived from a password in a known way, can use large amounts of CPU power to guess passwords, and try to match the piece of information.

How Long Should a Password Be?

- Anderson's Formula[1] gives us a tool with which to calculate the minimum size of a password:
  - Let:
    - P be the probability that an attack guesses a password in a specified period of time.
    - 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 total number of possible passwords
  - Then:

$$P \geq \frac{TG}{N}$$

[1]J. Anderson, "Information Security in a Multi-User Computer Environment", in Morris Rubinoff (ed.), *Advances in Computers* 12, Academic Press, New York (1972)

How Long Should a Password Be?

- Let's look at how secure your 4-digit PIN for your ATM card might be:
  - The number of possible passwords (PINs) is $N = 10^4$ (assuming that the digits 0-9 are allowed in each of the 4 positions in the PIN)
  - Assume that an attacker can make $G = 10{,}000$ guesses per second in an off-line attack
    - (Not an unreasonable number, given only 4 digits and a reasonably powerful machine.)
  - How long would it take to guess a PIN with absolute certainty?

$$P \geq \frac{TG}{N}, \text{ or, } T \leq \frac{PN}{G} = \frac{1.0 \times 10000}{10000} = 1\text{sec}$$

Scary?

How Long Should a Password Be?

- If we assume an on-line attack:
  - Let:
    - R be the communication line speed in characters/minute
    - E be the number of characters exchanged for a single login attempt
    - S be the length of the password, in characters
    - A be the number of letters in the alphabet from which password characters are obtained
    - The number of possible passwords (assuming they are all the same length) is $N = A^S$
    - The number of guesses per minute is $G = R/E$
    - The period of guessing extends over M months, so $T = (60*24*30)*M = 43200*M$
  - Then:

$$P \geq \frac{TG}{N} = \frac{43200 \times M\left(\frac{R}{E}\right)}{A^S} \quad \text{or} \quad A^S = 43200 \times \frac{MR}{PE} \quad \text{(Anderson's Formula)}$$

How Long Should a Password Be?

- Let's assume, for an on-line or off-line attack:
  - Passwords comprise letters from an alphabet of $A = 96$ characters
  - We can make $G = 10{,}000$ guesses every second
  - The period of guessing is a year ($T = 365$ days $= 365*24*60*60$ seconds).
- What is the minimum password length to achieve a probability of 0.5 of guessing the correct password?

$$P \geq \frac{TG}{N}, \quad \text{so} \quad N \geq \frac{TG}{P} = \frac{(365 \times 24 \times 60 \times 60) \times 10000}{0.5} \approx 6.3 \times 10^{11}$$

$$N = A^S \geq 6.3 \times 10^{11}$$

$$\text{so:} \quad S \geq \log_A(6.3 \times 10^{11}) = \frac{\log_{10}(6.3 \times 10^{11})}{\log_{10} A} = \frac{11.8}{1.98} \approx 6 \text{ characters}$$

How Long Should a Password Be?

- However, the previous analysis *assumed random password choices*. This is far from the actual case in the real world:
  - People tend to:
    - Choose minimum length passwords
    - Choose real words as passwords
    - Choose the name of their mother/father/child/brother/sister/dog/cat
    - Be annoyed by/resistant to security

Password Guidelines

- Here's a compendium of hints:
  - Choose a password that is easy to remember and difficult to guess (Duh!)
  - Never share your password (Duh!)
  - Never write down your password (Duh!)
  - Choose a long password
  - Use a combination of letters, numbers (0-9), and standard symbols (! @ # $ % ^ & *) (If that's allowed by your system)
  - Don't use personal information that someone could easily figure out, such as your birthday, your own/spouse's/child's/parent's/sibling's name, or phone number.
  - Avoid obvious passwords such as "123456", "test", "password", your login name, etc.
  - Consider using a automatic password generator (however, see later...)

Password Guidelines

- Do ***not*** choose a password that is:
  - ⇒ A word in *any* dictionary
  - ⇒ Anyone's name (there are lots of proper name lists out there)
  - ⇒ A word in any "cracking dictionary": lists of words that crackers use to try to crack passwords. These include:
    - » Abbreviations, asteroids, biology, cartoons, character patterns, machine names, famous names, female names, Bible names, male names, movies, myths-legends, number patterns, short phrases, placenames, Science Fiction, Shakespeare, songs, sports, surnames
  - ⇒ Acronyms, or product names
  - ⇒ Names from popular culture ("Spock", "JayLo")
  - ⇒ Concatenations of any of the above
  - ⇒ Any of the above, with a single character preceding or following it ("9adam", "goofy1")
  - ⇒ Any of the above, reversed ("jane" --> "enaj"), doubled ("ben" --> "benben") or mirrored ("john" --> "johnnhoj")
  - ⇒ Any of the above, with a zero (0) substituted for the letter o, or a one (1) substituted for the letter l (ell).
  - ⇒ Any of the above with all vowels and/or white spaces deleted
  - ⇒ Words like "foobar", "snafu", "xyzzy", and "qwerty" (and other keyboard patterns)
  - ⇒ All numerals, such as your license plate number, or social security number

Password Guidelines

- ***Do*** choose a password that is:
  - ⇒ Something easy to remember with at least 6 (preferably 8 or more) characters with a mix of alphabetic and numeric characters (spaces aren't usually allowed)
  - ⇒ Something obscure
    - » A deliberately misspelled term or using an odd character in an otherwise familiar term, such as `phnybon` instead of `funnybone`, or `vege%tarian`
    - » A combination of two otherwise unrelated words, such as `ferritemonkey`
  - ⇒ A combination of letters and numbers, or a phrase such as `mnYc0l0rz` for "`many colors`", or a misspelled phrase such as `2HotPeetzas` or `ItzAGurl`.
  - ⇒ An acronym for your favorite saying or quote. For example, `Whrzzbf` ("`Where's the beef?`") or `L!isn!` ("`Live! It's Saturday Night!`").
  - ⇒ An easily pronounceable nonsense word, such as `bargleFlatz` or `bitz-lunk-flob`
  - ⇒ Two words separated by a non-alphabetic, non-numeric, or punctuation character, for example: `beef2%stone` or `bone,poke#`
  - ⇒ Something that no one but you would ever think of. The best password is one that is totally random to anyone else except you. It is difficult to tell you how to come up with these, but people are able to do it. Use your imagination!

Password Guidelines

- Another definite No-No is to use the default password for an account provided by a system, without changing it.
- You can assume that password crackers will have a list of such default passwords, and will try those immediately
- In fact, there are databases of vendor default passwords publicly available on the web. Here's one:

  `http://www.cirt.net/cgi-bin/passwd.pl`

Automatically Generating Passwords

- One approach taken by some administrators is to generate passwords for users, and not allow them to choose their own.
  - Sometimes these passwords are totally random, unpronounceable, strings such as `fgqxx&@56dlg7!`
  - Sometimes they are pronounceable nonsense words, such as `helbothnock` or `gladgorargs`
- The assumption is that a randomly generated password is automatically more secure, but this may not always be the case:

Automatically Generating Passwords

- Morris and Thompson[1] document a PDP-11 system that randomly generated passwords composed of 8 capital letters and digits.
  - Theoretically, this implied that the number of possible passwords was $(26 + 10)^8 = 36^8 = 2821109907456 = 2.82 \times 10^{12}$
  - Assuming that an attacker could try 1000 guesses per second, how long would it take to crack a password?

$$P \geq \frac{TG}{N}, \text{ or, } T \leq \frac{PN}{G} = \frac{1.0 \times 2.82 \times 10^{12}}{1000} \text{sec}$$

$$= 2.82 \times 10^9 \text{ sec} = \frac{2.82 \times 10^9}{60 \times 60 \times 24 \times 365} = 89 \text{years}$$

[1]R. Morris and K. Thompson, "Password Security: A Case History", *Communications of the ACM*, 22 (11), pp 594-597 (Nov 1979)

Automatically Generating Passwords

- However, an attacker noticed that the pseudorandom number generator (PRNG) was run on the PDP-11, which was a 16-bit machine. As a result, the PRNG had a period of $2^{16} - 1 = 65{,}535$ possible passwords.
- Thus:

$$P \geq \frac{TG}{N}, \text{ or, } T \leq \frac{PN}{G} = \frac{1.0 \times 65535}{1000} \text{sec}$$

$$= 65.5 \text{sec}$$

- They actually took only 41 seconds to find all the passwords.
- The lesson here is that you truly have to analyze the system to determine its true security level.

Password Administrator Guidelines

- It's common for administrators to try to force users to become more security conscious:
  - Make them choose long passwords
  - Force them to change their password on a frequent basis
  - Prevent them from changing their password to the same password they were just using (or any of their previous *n* passwords, or a password that looks "too much like" one of those)
  - Generate randomized passwords for users, refusing to allow them to choose.
- It's important for administrators to realize that such activities can rapidly become self-defeating when users simply can't remember their passwords, and end up writing them down -- sometimes in public areas.
  - That's ***less*** secure, not more!

Password Administrator Guidelines

- As the textbook (page 247) says:

  "In general, it's impossible to make systems secure without the cooperation of legitimate users. If frequent password changes address a real threat, users must be educated to fear that threat so they will strive for good passwords. If the threat cannot be made real to the users, the inconvenience won't be worth the trouble."

and on page 245:

"The classic tale is one where the system manager's password is posted on the console because it is too long and complex to remember."

Password Guidelines for Multiple Systems

- The common wisdom is to tell users to use different passwords on different systems they have access to.
- The reason is obvious:
  - ***If one user account becomes compromised, they all are.***
- However, this must be balanced by the reality that users can't remember lots of different passwords (and typically won't put up with a lot of hassle), and so will write the passwords down.

Single Sign-On

- How many of you have multiple accounts (Rivier, your own ISP, email accounts, web sites, etc.) ?
- Is it difficult to remember them all? Do you try to make them all the same, or do you make an effort to make them all different?
- Wouldn't it be nice if you could simply have a single user id and password for all of them?
- This is called ***Single Sign-On (SSO),*** and requires the use of a ***single sign-on system***.
  - ***Microsoft Passport*** comes to mind
    - But see http://avirubin.com/passport.html
  - as does the competitive ***Liberty Alliance Project***,
  - and the attempt to standardize SSO

Single Sign-On

- Ferguson & Schneier, in *Practical Cryptography* (op cit.) say:

  "In practice, this just plain doesn't work. There is no widely-used standard for this process, and until there is, it won't happen automatically. Just think of all the different applications what would have to be changed to automatically get their passwords from the single sign-on system."
- Perhaps this is slightly out of date (the book was published in 2003), but it certainly is clear that things aren't changing overnight – especially since there are competing systems being created.
- They suggest instead the use of a small program to store all the usernames and passwords in a text file. Schneier has written a public domain program, Password Safe, (which uses the BlowFish cipher) to do this.

Storage of Passwords on Systems

- When your system administrator sets your password, what is stored on your computer?
  - Originally, the password was stored in plaintext, but the password file was protected.
  - This was considered insufficient, so systems developed one-way hash functions:
    - The password is hashed, and the result stored in the password file.
    - Sometimes (on some UNIX systems) the resulting password file is allowed to be read-accessible.
      - This is not a good idea, because it is susceptible to off-line password attacks.
    - When a user logs in and supplies a password, it is hashed using the same one-way hash function, and the result compared with the hash stored in the password file. If they match, the login is successful; otherwise, not.

Storage of Passwords on Systems

- On many UNIX systems, the following algorithm is used:
  1) A user selects a password of up to 8 characters (the remaining characters are ignored)
  2) This is converted into a 56-bit value (using 7-bit ASCII) which is then input into an encryption function, known as **`crypt (3)`**.
  3) The function uses the DES algorithm, which is modified using a 12-bit randomly generated "***salt***" value.
  4) The algorithm is repeated for a total of 25 encryptions
  5) The resulting 64-bit output is then translated into an 11-character sequence
  6) This 11-character value is stored, along with a plaintext copy of the salt, in the password file entry for that user.
- The salt does the following:
  - Prevents duplicate passwords from being visible in the password file.
  - Effectively increases the length of the password over what the user could do
  - Prevents the use of hardware implementations of DES in an attack

Dictionary Attacks

- A very successful off-line password guessing attack is the ***Dictionary Attack***, where a dictionary (usually enhanced) is used.
  - Since the target machine is not involved (it's an off-line attack), evern word in a 50,000-word dictionary can be tested in only a minute.
- Klein[1] collected approximately 15,000 encrypted passwords from UNIX password files, and used a cracker program, based on a dictionary and simple assumptions.
  - Here is an excerpt from his results:

    "... 21% (nearly 3,000 passwords) were guessed in the first week, and that in the first 15 minutes of testing, 368 passwords (or 2.7%) had been cracked using what experience has shown would be the most fruitful line of attack (i.e., using the user or account names as passwords)."

[1]Klein, D., "Foiling the Cracker: A Survey of, and Improvements to, Password Security", Proceedings, *UNIX Security Workshop II*, August 1990

Dictionary Attacks

- Here are some of Klein's results:

Type of Password	Percentage of Passwords Matched
User/account name	2.7
Character sequences	0.2
Numbers	0.1
Chinese	0.4
Place names	0.6
Common names	4.0
Female names	1.2
Male names	1.0
Uncommon names	0.9
Myths and legends	0.5
Shakespearean	0.1
Sports terms	0.2
Science Fiction	0.4
Movies and actors	0.1
Cartoons	0.1
Famous people	0.4
Phrases and patterns	1.8
Surnames	0.1
Biology	0.0
System dictionary	7.4
Machine names	1.0
Mnemonics	0.0
King James Bible	0.6
Miscellaneous words	0.4
Yiddish words	0.0
Asteroids	0.1
Total	**24.3**

Password Cracking Tools

- Here are some password cracking tools available on the Internet:
  - ***l0phtcrack***, a Windows NT/2000 password cracker
    - http://www.whitehatinc.com/w2ktools/l0phtcrack/
  - ***John the Ripper***, a password cracker for UNIX, DOS, Win32, BeOS, and OpenVMS
    - http://www.openwall.com/john/
  - ***Crack***, one of the earliest powerful UNIX password cracking tools
    - http://www.crypticide.org/users/alecm/
  - ***Pandora***, a tool for testing Novell Netware, including password cracking
    - http://www.nmrc.org/project/pandora/index.html
  - ***PalmCrack***, a tool for cracking Windows NT and UNIX passwords; runs on the PalmOS PDA platform
    - http://www.noncon.org/noncon/download.html

Trojan Horse Password Capture

- It's relatively easy to write a program that can fool people into thinking they are interacting with the operating system, when in fact, the program is imitating the O/S.
- I programmed the dialog on the right in a very short amount of time.
- Someone with more time and more determination could do a much better job.

Can anyone detect the dead giveaway that it isn't a real Microsoft login dialog?

Address-Based Authentication

- Assumes that the identity of the source can be inferred based on the source network address of the message packets.
- Supported on UNIX and OpenVMS
- Concept:
  - Each computer stores information which specifies the accounts on other computers that should have access to its resources.
  - Requests for resources include:
    - Copying files, logging in, executing a command remotely
- On UNIX, the Berkeley rtools support this style of access
- On OpenVMS, the equivalent functionality is known as a ***proxy***.

Address-Based Authentication

- On UNIX, there are two account mapping schemes:
  - **`/etc/hosts.equiv`** -- a global file which contains a list of computers which have identical user account assignments.
  - **`.rhosts`** file, in each UNIX user's home directory, which contains a list of (computer, account) pairs that are allowed access to the user's account.
- On OpenVMS, normal users are not allowed to control their own proxy access.
  - Instead, there is a centrally-managed proxy database that specifies for each (computer, account) pair what account(s) that pair may access.

Address-Based Authentication

- Address-based authentication is subject to two threats:
  - Once an intruder has gained access to an account, s/he has also gained access to all that account's proxy accounts on other machines.
    - How the intruder determines which hosts have proxy accounts for this account is an interesting question.
  - If an intruder can impersonate network addresses of nodes, s/he can access all the network resources of all users who have accounts on those nodes.
- Depending on the environment, address-based authentication may be more or less secure than sending passwords in the clear.
- But address-based authentication is highly convenient, and is very commonly used in many distributed environments.

Authentication Using Public Key Cryptography

- All the precautions of creating a decent password for a user can be for naught:
  - A user tries to log into a remote machine, and in the course of the challenge/response, the user's password gets sent to the remote machine in plaintext. Anyone eavesdropping on the line can read the password.
- One solution is the use of public key cryptography:
  1) Alice sends a message to Bob (the remote computer) "I'm Alice"
  2) Bob picks a random value, R, and sends it back to Alice.
  3) Alice encrypts R with her private key, KR_A, and sends it back to Bob.
  4) Bob decrypts the message with Alice's public key, KU_A, and checks that the resulting R and the original R match.

Authentication Using Secret Key Cryptography

- If we use symmetric (private key) cryptography, we can use the following protocol:
  1) Alice and Bob share a secret
  2) Alice sends a message to Bob "I'm Alice"
  3) Bob picks a random value, R, and sends it back to Alice
  4) Alice computes *CryptographicFunction*(*Shared secret* || R), and sends it back to Bob.
  5) Bob computes the same cryptographic function, and compares the the result to the message Alice sent, to see if they match.

Eavesdropping and Server Database Reading

- Public key cryptography can protect against these two attacks:
  - Eve, eavesdropping on the communication line
    - Nothing useful can be read by Eve, because the critical stuff is encrypted.
  - Eve, breaking into Bob (the computer), and finding Alice's key in Bob's database
    - Bob's database only holds Alice's public key, so it's not useful to Eve.
- Private (symmetric key) cryptography can also be used:
  - Lamport's Hash protocol (see the textbook for details)

Trusted Intermediaries

- We have seen earlier that secret key distribution is extremely problematic, and gets worse -- quickly! -- as the number of users /computers increases.
  - The assumption here is that, in a network of n computers, each computer needs to be able to authenticate every other computer, so each computer would need to know $(n - 1)$ keys, so we have a total of $n(n - 1) = (n^2 - n)$ keys distributed.
  - If we add a new computer, then we need to distribute additional keys:
    - n -- one to each of the existing computers, for the new machine, plus:
    - n -- one to the new machine, for each of the existing computers,
    - For a total of:

      $(n + 1)n = n^2 + n = n(n - 1) + 2n$

This gets unmanageable, rapidly!

Key Distribution Centers (KDC)

- To avoid this rapid expansion problem, we can delegate the job of managing keys to a ***trusted intermediary node***, known as a ***Key Distribution Center***.
  - The KDC knows keys for all the nodes.
  - If a new machine is added to the network, the only keys needing to be distributed are:
    - One for the new node (which needs to know its own key), plus
    - One for the KDC (the new node's key)
    - For a total of 2 (regardless of how many nodes there are in the entire network)

Key Distribution Centers (KDCs)

- If node Fred wants to talk to node Mary, then:

  1) Fred talks to the KDC (securely, using the key Fred shares with the KDC), and asks for the key to use to talk to node Mary.

  2) The KDC chooses a random number, R_{FM}, to be used as a key to be shared by Fred and Mary for their conversation.
     - It encrypts R_{FM} with the key that the KDC shares with Fred, and sends the result to Fred.
     - It encrypts R_{FM} with the key that the KDC shares with Mary, and sends the result to Mary (or sends it to Fred for Fred to send to Mary -- this is known as a ***ticket***, and often contains other information such as an expiration time, and Fred's name.)

  3) Fred then contacts Mary, and the two conduct a conversation using the information provided by the KDC.

Problems with KDCs

- KDCs have disadvantages:
  - If a KDC is ***compromised***, ***all the network resources are vulnerable***
  - The KDC is a ***single point of failure***. If it goes down, nobody can communicate any more.
  - The KDC could be a ***performance bottleneck***, because every node who wants to conduct a conversation with any other node must go through the KDC to start the conversation.
- ***Multiple, replicated, KDCs*** could alleviate the second two problems, but ***add complexity, cost***, ***and vulnerability***.

Certification Authorities (CAs)

- Key distribution is easy using public key cryptography, with public keys published in some known place.
- However, there is still the problem of Eve posting a bogus public key purporting to be Alice's, and thereby impersonating Alice.
- The typical solution is to have a trusted node known as a ***Certification Authority (CA)***
  - The ***CA generates certificates*** (signed messages specifying Alice's name, and her public key).
  - All nodes need to know the CA's public key so they can verify its signature on the certificates.

Certification Authorities (CAs)

- CAs are the public key equivalent of KDCs; if a CA is compromised, it can destroy the integrity of the entire network, just like a KDC.
- Advantages of CAs include:
  - The CA does not need to be on-line at all times.
  - As a result, it can be a much simpler device, and simpler often means more secure.
  - If a CA crashes, it merely means that new users cannot be installed, nor existing users modified. This means that multiple replicated CAs aren't as necessary.
  - Certificates are not security-sensitive, since they cannot be created or tampered with (although they could be deleted)
  - A compromised CA cannot decrypt conversations (unlike with KDCs); it's bad, but not as bad as a compromised KDC.

Certificate Revocation

- We don't necessarily want to create certificates that are valid forever, so certificates commonly have expiration dates, after which they become invalid.
- To handle the case of the disgruntled former employee who has a certificate, a CA maintains a ***Certificate Revocation List (CRL)***
  - Analogous to list of stolen or invalid credit cards.

Certificate Revocation

- The *ISO X.509* defined format for a certificate, part of the ***Public Key Infrastructure (PKI)***, also has a defined CRL.
- A CRL lists serial numbers for certificates that should not be honored.
- A new CRL is posted periodically, and lists all revoked and unexpired certificates.
- A certificate is valid if:
  - it has a valid CA signature
  - has not expired, and
  - is not listed in the CA's most recent CRL.

Multiple Trusted Intermediaries

- If we are dealing with a Wide Area Network (WAN), we immediately encounter the question:

  Who is the single administration trusted by all principals[1]?

- On an international scale (or even between different organizations), the answer is:

Absolutely no one!

[1]**Principal**: Anything or anyone participating in a secure communication

Multiple Trusted Intermediaries

- The solution is to break the world into ***domains***
- Each domain has its own trusted administration (KDC or CA)
  - The textbook uses examples of KDCs for the CIA and KGB domains.
  - A person in one domain who wishes to authenticate someone in another domain must work through multiple KDCs/Cas, which in turn communicate with each other across the domains.

do·main
1 a : complete and absolute ownership of land **b** : land so owned
2 : a territory over which dominion is exercised
3 : a region distinctively marked by some physical feature <the *domain* of rushing streams, tall trees, and lakes>
4 : a sphere of knowledge, influence, or activity <the *domain* of art>
5 : the set of elements to which a mathematical or logical variable is limited; *specifically* : the set on which a function is defined
6 : any of the small randomly oriented regions of uniform magnetization in a ferromagnetic substance

Session Key Establishment

- Once Alice and Bob have authenticated each other, and then wish to continue their conversation privately, they must find some way of communicating in other than plaintext.
- They could use a shared secret for every session they conduct, but this can be compromised because Eve can obtain a large number of encrypted messages and use this additional information to break the cipher.
- It is much better for Alice and Bob to decide on a different key for each session.
- One such protocol is the ***Secure Sockets Layer (SSL)***:
  - SSL v2 was originally developed by Netscape (deployed in Navigator 1.1) in 1995
  - Microsoft fixed a number of security problems and introduced Private Communications Technology (PCT)
  - Netscape overhauled the protocol as SSLv3
  - The IETF then introduced a similar but incompatible protocol, ***Transport Layer Security (TLS)***

Authentication Tokens

- Something that a person ***has***.
  - A physical device that a person carries and uses to authenticate him/herself.
    - ***Key*** for home, office, car, etc.
    - ***Credit card*** -- magnetic strip, so requires custom hardware
    - ***Smart card*** -- about the size of a credit card, but with an embedded CPU and memory; when inserted in a ***smart card reader***, carries on a conversation with the device.
      - PIN protected memory card
      - Cryptographic challenge/response card
      - Cryptographic calculator (a.k.a. Readerless smart card)

Biometric Devices

- Technology available today includes:
  - Retinal scanner
  - Fingerprint reader
  - Face recognition
  - Iris scanner
  - Handprint reader
  - Voiceprint
  - Keystroke timing
  - Signature
- None is really terribly viable, but things are changing as we speak...

Authentication Protocols

- The textbook has two entire chapters on *"Security Handshake Pitfalls"* and *"Strong Password Protocols"*, which describe a number of authentication protocols.
- I'm not going to go into the details here -- you can read the chapter as well as I can.
- I'll just point out some highlights...

Authentication Protocols

- In *"Mediated Authentication (with KDC)"*, p. 274ff, the textbook describes the concepts and introduces a classic authentication protocol with KDCs:
  - The Needham-Schroeder Protocol

Kerberos

M12.1 Kerberos & Herakles

KERBEROS was the fierce watchdog of Hades. He was depicted as a three-headed dog with a serpent's tail, a mane of snakes, and a lion's claws.

- ***Kerberos*** is a secret key based service for providing authentication in a network
  - Many modern systems base their authentication on Kerberos
  - It was originally designed at MIT, and was based on work by Needham and Schroeder.
  - The textbook has two entire chapters on Kerberos: *"Kerberos V4"* and *"Kerberos V5"*, both of which go into a fair degree of detail about the two implementations.

Summary

- Well, that's a fair amount on the subject of authentication
  - There's lots more, but we have to stop somewhere.
  - I hope that this provided a balance between practical ideas (password selection, etc.) and some of the more formal authentication protocol ideas.
  - Onward!