CS162
Operating Systems and Systems Programming
Lecture 25

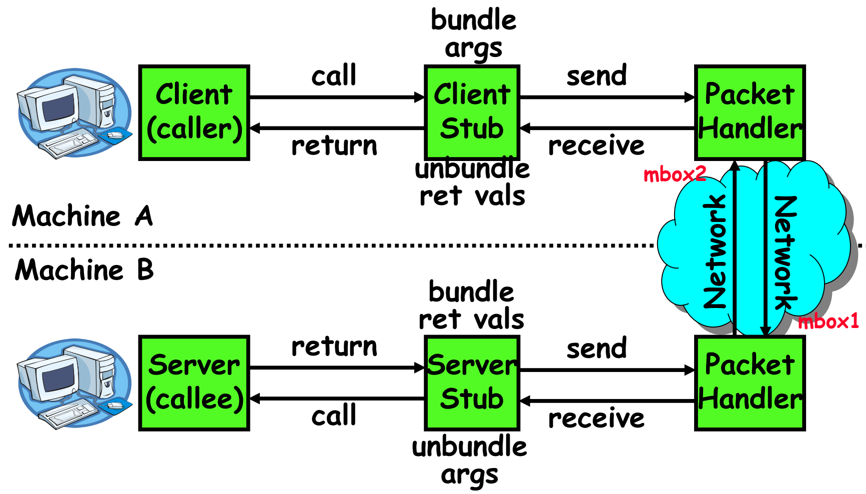
Protection and Security in Distributed Systems

November 28, 2005

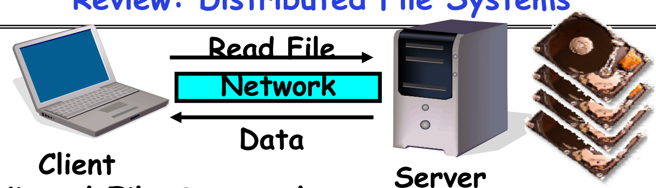
Prof. John Kubiatowicz

http://inst.eecs.berkeley.edu/~cs162

Review: RPC Information Flow



Review: Distributed File Systems



- **VFS:** Virtual File System layer
  - Provides mechanism which gives same system call interface for different types of file systems
- **Distributed File System:**
  - Transparent access to files stored on a remote disk
    - » NFS: Network File System
    - » AFS: Andrew File System
  - Caching for performance
- **Cache Consistency:** Keeping contents of client caches consistent with one another
  - If multiple clients, some reading and some writing, how do stale cached copies get updated?
  - NFS: check periodically for changes
  - AFS: clients register callbacks so can be notified by server of changes

Goals for Today

- Security Mechanisms
  - Authentication
  - Authorization
  - Enforcement
- Cryptographic Mechanisms

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne

Protection vs Security

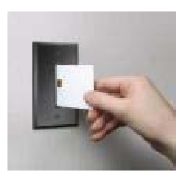
- **Protection:** one or more mechanisms for controlling the access of programs, processes, or users to resources
  - Page Table Mechanism
  - File Access Mechanism
- **Security:** use of protection mechanisms to prevent misuse of resources
  - Misuse defined with respect to policy
    - » E.g.: prevent exposure of certain sensitive information
    - » E.g.: prevent unauthorized modification/deletion of data
  - Requires consideration of the external environment within which the system operates
    - » Most well-constructed system cannot protect information if user accidentally reveals password
- What we hope to gain today and next time
  - Conceptual understanding of how to make systems secure
  - Some examples, to illustrate why providing security is really hard in practice

Preventing Misuse

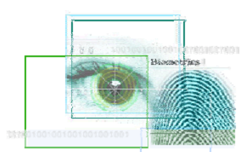
- Types of Misuse:
  - Accidental:
    - » If I delete shell, can't log in to fix it!
    - » Could make it more difficult by asking: "do you really want to delete the shell?"
  - Intentional:
    - » Some high school brat who can't get a date, so instead he transfers $3 billion from B to A.
    - » Doesn't help to ask if they want to do it (of course!)
- Three Pieces to Security
  - **Authentication:** who the user actually is
  - **Authorization:** who is allowed to do what
  - **Enforcement:** make sure people do only what they are supposed to do
- Loopholes in any carefully constructed system:
  - Log in as superuser and you've circumvented authentication
  - Log in as self and can do anything with your resources; for instance: run program that erases all of your files
  - Can you trust software to correctly enforce Authentication and Authorization?????

Authentication: Identifying Users

- How to identify users to the system?
  - Passwords
    - » Shared secret between two parties
    - » Since only user knows password, someone types correct password ⇒ must be user typing it
    - » Very common technique
  - Smart Cards
    - » Electronics embedded in card capable of providing long passwords or satisfying challenge → response queries
    - » May have display to allow reading of password
    - » Or can be plugged in directly; several credit cards now in this category



  - Biometrics
    - » Use of one or more intrinsic physical or behavioral traits to identify someone
    - » Examples: fingerprint reader, palm reader, retinal scan
    - » Becoming quite a bit more common



Passwords: Secrecy

- System must keep copy of secret to check against passwords
  - What if malicious user gains access to list of passwords?
    - » Need to obscure information somehow
  - Mechanism: utilize a transformation that is difficult to reverse without the right key (e.g. encryption)
- Example: UNIX /etc/passwd file
  - passwd→one way transform(hash)→encrypted passwd
  - System stores only encrypted version, so OK even if someone reads the file!
  - When you type in your password, system compares encrypted version
- Problem: Can you trust encryption algorithm?
  - Example: one algorithm thought safe had back door
    - » Governments want back door so they can snoop
  - Also, security through obscurity doesn't work
    - » GSM encryption algorithm was secret; accidentally released; Berkeley grad students cracked in a few hours

Administrivia

- My office hours
  - No office hours Thursday (Thanksgiving)
- Project 4 design document
  - Due Tomorrow (November 29th)
- MIDTERM II: Monday December 5th!
  - 5:30-8:30pm, 10 Evans
  - All material from last midterm and up to previous class
  - Includes virtual memory
- Review Session:
  - Thursday evening 6-8pm
  - Location: 50 Birge
- Final Exam
  - December 17th, 12:30 – 3:30, 220 Hearst Gym
- Final Topics: Any suggestions?

Passwords: How easy to guess?

- Ways of Compromising Passwords
  - Password Guessing:
    - » Often people use obvious information like birthday, favorite color, girlfriend's name, etc…
  - Dictionary Attack:
    - » Work way through dictionary and compare encrypted version of dictionary words with entries in /etc/passwd
  - Dumpster Diving:
    - » Find pieces of paper with passwords written on them
    - » (Also used to get social-security numbers, etc)
- Paradox:
  - Short passwords are easy to crack
  - Long ones, people write down!
- Technology means we have to use longer passwords
  - UNIX initially required lowercase, 5-letter passwords: total of 26^5=10million passwords
    - » In 1975, 10ms to check a password→1 day to crack
    - » In 2005, .01μs to check a password→0.1 seconds to crack
  - Takes less time to check for all words in the dictionary!

Passwords: Making harder to crack

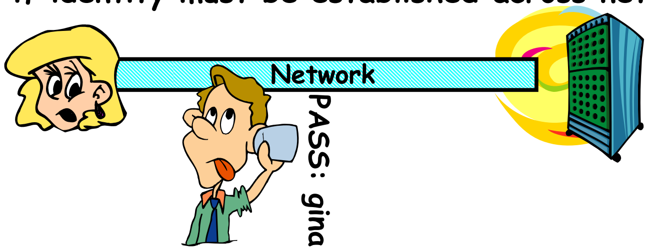
- How can we make passwords harder to crack?
  - Can't make it impossible, but can help
- Technique 1: Extend everyone's password with a unique number (stored in password file)
  - Called "salt". UNIX uses 12-bit "salt", making dictionary attacks 4096 times harder
  - Without salt, would be possible to pre-compute all the words in the dictionary hashed with the UNIX algorithm: would make comparing with /etc/passwd easy!
  - Also, way that salt is combined with password designed to frustrate use of off-the-shelf DES hardware
- Technique 2: Require more complex passwords
  - Make people use at least 8-character passwords with upper-case, lower-case, and numbersd
    - » 70^8=6x10^{14}=6million seconds=69 days@0.01μs/check
  - Unfortunately, people still pick common patterns
    - » e.g. Capitalize first letter of common word, add one digit

Passwords: Making harder to crack (con't)

- Technique 3: Delay checking of passwords
  - If attacker doesn't have access to /etc/passwd, delay every remote login attempt by 1 second
  - Makes it infeasible for rapid-fire dictionary attack
- Technique 4: Assign very long passwords
  - Long passwords or pass-phrases can have more entropy (randomness→harder to crack)
  - Give everyone a smart card (or ATM card) to carry around to remember password
    - » Requires physical theft to steal password
    - » Can require PIN from user before authenticates self
  - Better: have smartcard generate pseudorandom number
    - » Client and server share initial seed
    - » Each login attempt advances to next random number
- Technique 5: "Zero-Knowledge Proof"
  - Require a series of challenge-response questions
    - » Distribute secret algorithm to user
    - » Server presents a number, say "5"; user computes something from the number and returns answer to server
    - » Server never asks same "question" twice
  - Often performed by smardcard plugged into system

Authentication in Distributed Systems

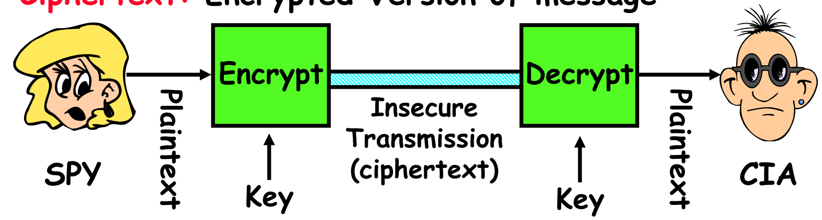
- What if identity must be established across network?



  - Need way to prevent exposure of information while still proving identity to remote system
  - Many of the original UNIX tools sent passwords over the wire "in clear text"
    - » E.g.: telnet, ftp, yp (yellow pages, for distributed login)
    - » Result: Snooping programs widespread
- What do we need? Cannot rely on physical security!
  - Encryption: Privacy, restrict receivers
  - Authentication: Remote Authenticity, restrict senders

Private Key Cryptography

- Private Key (Symmetric) Encryption:
  - Single key used for both encryption and decryption
- Plaintext: Unencrypted Version of message
- Ciphertext: Encrypted Version of message

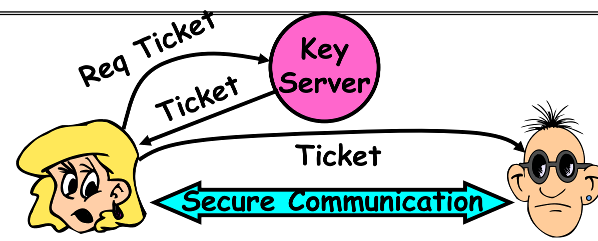


- Important properties
  - Can't derive plain text from ciphertext (decode) without access to key
  - Can't derive key from plain text and ciphertext
  - As long as password stays secret, get both secrecy and authentication
- Symmetric Key Algorithms: DES, Triple-DES, AES

Key Distribution

- How do you get shared secret to both places?
  - For instance: how do you send authenticated, secret mail to someone who you have never met?
  - Must negotiate key over private channel
    - » Exchange code book
    - » Key cards/memory stick/others
- Third Party: Authentication Server (like Kerberos)
  - Notation:
    - » K_{xy} is key for talking between x and y
    - » $(\ldots)^K$ means encrypt message (…) with the key K
    - » Clients: A and B, Authentication server S
  - A asks server for key:
    - » A→S: [Hi! I'd like a key for talking between A and B]
    - » Not encrypted. Others can find out if A and B are talking
  - Server returns *session* key encrypted using B's key
    - » S→A: Message [Use K_{ab} (This is A! Use K_{ab})^{Ksb}]^{Ksa}
    - » This allows A to know, "S said use this key"
  - Whenever A wants to talk with B
    - » A→B: Ticket [This is A! Use K_{ab}]^{Ksb}
    - » Now, B knows that K_{ab} is sanctioned by S

Authentication Server Continued



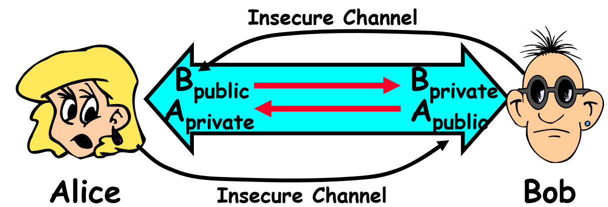
- Details
  - Both A and B use passwords (shared with key server) to decrypt return from key servers
  - Add in timestamps to limit how long tickets will be used to prevent attacker from replaying messages later
  - Also have to include encrypted checksums (hashed version of message) to prevent malicious user from inserting things into messages/changing messages
  - Want to minimize # times A types in password
    - » A→S (Give me temporary secret)
    - » S→A (Use $K_{temp-sa}$ for next 8 hours)^{Ksa}
    - » Can now use $K_{temp-sa}$ in place of K_{sa} in prototcol

Public Key Encryption

- Can we perform key distribution without an authentication server?
  - Yes. Use a Public-Key Cryptosystem.
- Public Key Details
  - Don't have one key, have two: K_{public}, $K_{private}$
    - » Two keys are mathematically related to one another
    - » Really hard to derive K_{public} from $K_{private}$ and vice versa
  - Forward encryption:
    - » Encrypt: $(\text{cleartext})^{K_{public}} = \text{ciphertext}_1$
    - » Decrypt: $(\text{ciphertext}_1)^{K_{private}} = \text{cleartext}$
  - Reverse encryption:
    - » Encrypt: $(\text{cleartext})^{K_{private}} = \text{ciphertext}_2$
    - » Decrypt: $(\text{ciphertext}_2)^{K_{public}} = \text{cleartext}$
  - Note that $\text{ciphertext}_1 \neq \text{ciphertext}_2$
    - » Can't derive one from the other!
- Public Key Examples:
  - RSA: Rivest, Shamir, and Adleman
    - » K_{public} of form (k_{public}, N), $K_{private}$ of form $(k_{private}, N)$
    - » N = pq. Can break code if know p and q
  - ECC: Elliptic Curve Cryptography

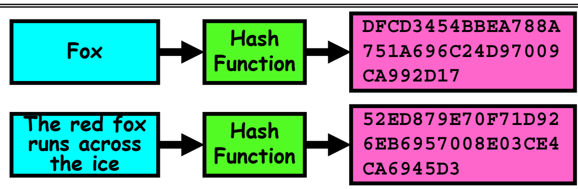
Public Key Encryption Details

- Idea: K_{public} can be made public, keep $K_{private}$ private



- Gives message privacy (restricted receiver):
  - Public keys (secure destination points) can be acquired by anyone/used by anyone
  - Only person with private key can decrypt message
- What about authentication?
  - Use combination of private and public key
  - Alice→Bob: $[(\text{I'm Alice})^{A_{private}} \text{ Rest of message}]^{B_{public}}$
  - Provides restricted sender and receiver
- But: how does Alice know that it was Bob who sent her B_{public}? And vice versa…

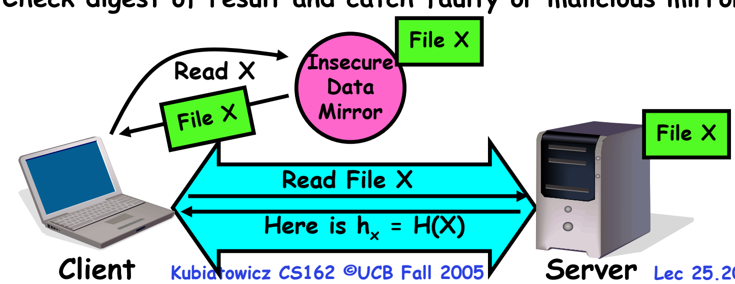
Secure Hash Function



- Hash Function: Short summary of data (message)
  - For instance, $h_1=H(M_1)$ is the hash of message M_1
    - » h_1 fixed length, despite size of message M_1.
    - » Often, h_1 is called the "digest" of M_1.
- Hash function H is considered secure if
  - It is infeasible to find M_2 with $h_1=H(M_2)$; ie. can't easily find other message with same digest as given message.
  - It is infeasible to locate two messages, m_1 and m_2, which "collide", i.e. for which $H(m_1) = H(m_2)$
  - A small change in a message changes many bits of digest/can't tell anything about message given its hash

Use of Hash Functions

- Several Standard Hash Functions:
  - MD5: 128-bit output
  - SHA-1: 160-bit output
- Can we use hashing to securely reduce load on server?
  - Yes. Use a series of insecure mirror servers (caches)
  - First, ask server for digest of desired file
    - » Use secure channel with server
  - Then ask mirror server for file
    - » Can be insecure channel
    - » Check digest of result and catch faulty or malicious mirrors



Signatures/Certificate Authorities

- Can use X_{public} for person X to define their identity
  - Presumably they are the only ones who know $X_{private}$.
  - Often, we think of X_{public} as a "principle" (user)
- Suppose we want X to sign message M?
  - Use private key to encrypt the digest, i.e. $H(M)^{Xprivate}$
  - Send both M and its signature:
    - » Signed message = $[M, H(M)^{Xprivate}]$
  - Now, anyone can verify that M was signed by X
    - » Simply decrypt the digest with X_{public}
    - » Verify that result matches H(M)
- Now: How do we know that the version of X_{public} that we have is really from X???
  - Answer: Certificate Authority
    - » Examples: Verisign, Entrust, Etc.
  - X goes to organization, presents identifying papers
    - » Organization signs X's key: $[X_{public}, H(X_{public})^{CAprivate}]$
    - » Called a "Certificate"
  - Before we use X_{public}, ask X for certificate verifying key
    - » Check that signature over X_{public} produced by trusted authority
- How do we get keys of certificate authority?
  - Compiled into your browser, for instance!

Conclusion

- User Identification
  - Passwords/Smart Cards/Biometrics
- Passwords
  - Encrypt them to help hid them
  - Force them to be longer/not amenable to dictionary attack
  - Use zero-knowledge request-response techniques
- Distributed identity
  - Use cryptography
- Symmetrical (or Private Key) Encryption
  - Single Key used to encode and decode
  - Introduces key-distribution problem
- Public-Key Encryption
  - Two keys: a public key and a private key
    - » Not derivable from one another
- Secure Hash Function
  - Used to summarize data
  - Hard to find another block of data with same hash