CS162
Operating Systems and Systems Programming
Lecture 26

Protection and Security in Distributed Systems II

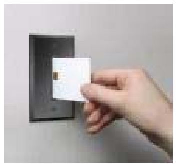
November 30, 2005

Prof. John Kubiatowicz

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

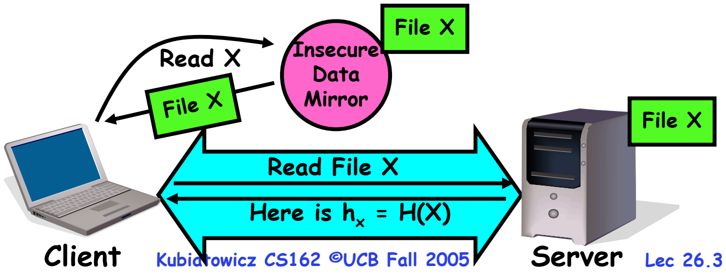
Review: Authentication: Identifying Users

- 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



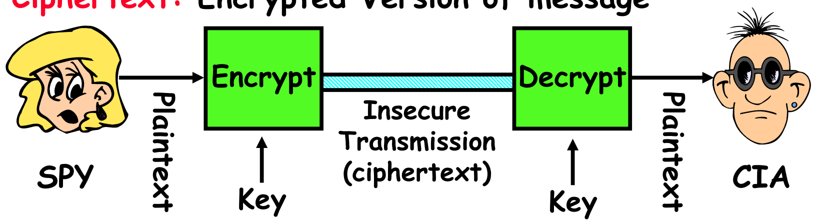
Review: Use of Hash Functions

- Let h_1=H(M_1); hash function H is considered secure if:
  - It is infeasible to find M_2 with h_1=H(M_2); i.e. 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)
- Can we use hashing to securely reduce load on server?
  - 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



Review: Private Key Cryptography

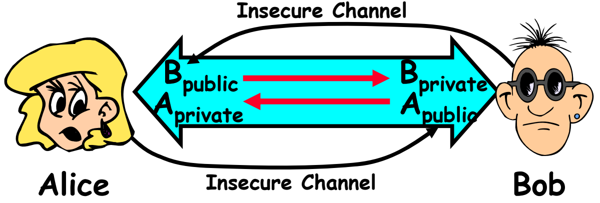
- Private Key (Symmetric) Encryption:
  - Single key used for both encryption and decryption
- Plaintext: Unencrypted Version of message
- Ciphertext: Encrypted Version of message



- 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

Review: Public Key Encryption Details

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



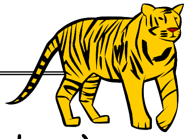
- What about authentication?
  - Use combination of private and public key
  - Alice→Bob: [(I'm Alice)^{Aprivate} Rest of message]^{Bpublic}
  - Provides restricted sender and receiver
- How does Alice know it was Bob who sent her B_{public}?
  - Answer: Certificate Authority
    - » Examples: Verisign, Entrust, Etc.
  - B goes to organization, presents identifying papers
    - » Organization signs B's key: [B_{public}, $H(B_{public})^{CAprivate}$]
  - Before we use B_{public}, ask B for certificate verifying key
    - » Check that signature over B_{public} produced by trusted authority

Goals for Today

- Use of Cryptographic Mechanisms
- Authorization Mechanisms
- Worms and Viruses

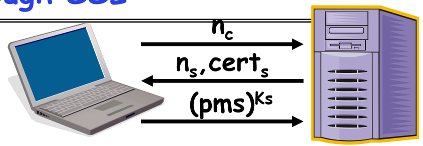
Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne

Cryptographic Summary



- Private Key Encryption (also Symmetric Key)
  - Pros: Very Fast
    - » can encrypt at network speed (even without hardware)
  - Cons: Need to distribute *secret* key to both parties
- Public Key Encryption (also Asymmetric Key)
  - Pros: Can distribute keys in public
    - » Although need some sort of certificate authority: Often called a Public Key Infrastructure (PKI)
  - Cons: Very Slow
    - » 100—1000 times slower than private key encryption
- Session Key
  - Randomly generated private key used for single session
  - Often distributed via public key encryption
- Secure Hash
  - Fixed length summary (digest) of data; security properties make it effectively hard to spoof
- Message Authentication Code (MAC)
  - Technique for using secure hash and session key to verify individual packets (even at the IP level)
- Signature over Document
  - Hash of document encrypted with private key

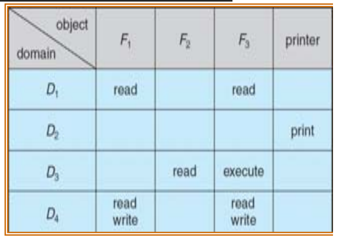
Security through SSL



- SSL Web Protocol
  - Port 443: secure http
  - Use of public-key encryption for key-distribution
- Server has a certificate signed by certificate authority
  - Contains server info (organization, IP address, etc)
  - Also contains server's public key and expiration date
- Establishment of Shared, 48-byte "master secret"
  - Client picks 28-byte random value n_c to server
  - Server returns its own 28-byte random value n_s, plus its certificate $cert_s$
  - Client verifies certificate by checking with public key of certificate authority compiled into browser
    - » Also check expiration date
  - Client picks 46-byte "premaster" secret (pms), encrypts it with public key of server, and sends to server
  - Now, both server and client have n_c, n_s, and pms
    - » Each can compute 48-byte master secret using one-way and collision-resistant function on three values
    - » Random "nonces" n_c and n_s make sure master secret fresh

Authorization: Who Can Do What?

- How do we decide who is authorized to do actions in the system?
- Access Control Matrix: contains all permissions in the system
  - Resources across top
    - » Files, Devices, etc…
  - Domains in columns
    - » A domain might be a user or a group of permissions
    - » E.g. above: User D_3 can read F_2 or execute F_3
  - In practice, table would be huge and sparse!



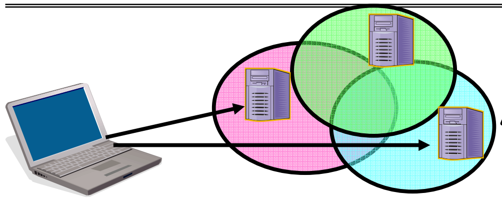
domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- Two approaches to implementation
  - Access Control Lists: store permissions with each object
    - » Still might be lots of users!
    - » UNIX limits each file to: r,w,x for owner, group, world
    - » More recent systems allow definition of groups of users and permissions for each group
  - Capability List: each process tracks objects has permission to touch
    - » Popular in the past, idea out of favor today
    - » Consider page table: Each process has list of pages it has access to, not each page has list of processes …

Administrivia

- MIDTERM II: Monday December 5th! (next Monday)
  - 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
  - Cover all topics of course

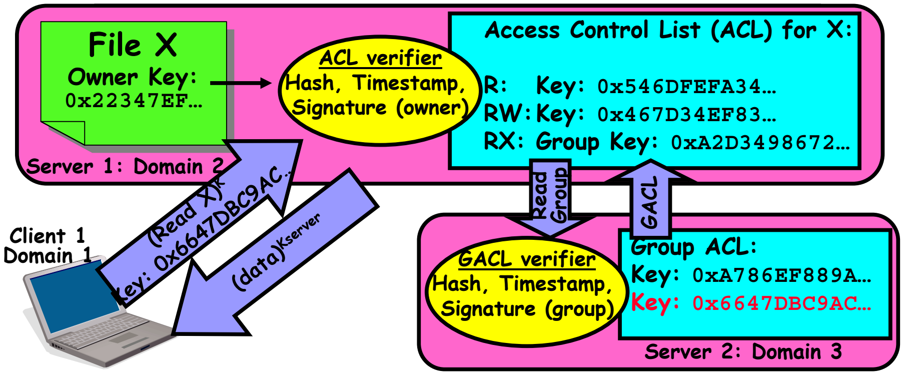
How to perform Authorization for Distributed Systems?



Different Authorization Domains

- Issues: Are all user names in world unique?
  - No! They only have small number of characters
    - » kubi@mit.edu → kubitron@lcs.mit.edu → kubitron@cs.berkeley.edu
    - » However, someone thought their friend was kubi@mit.edu and I got very private email intended for someone else…
  - Need something better, more unique to identify person
- Suppose want to connect with any server at any time?
  - Need an account on every machine! (possibly with different user name for each account)
  - OR: Need to use something more universal as identity
    - » Public Keys! (Called "Principles")
    - » People *are* their public keys

Distributed Access Control



File X

Owner Key: 0x22347EF…

ACL verifier

Hash, Timestamp, Signature (owner)

Access Control List (ACL) for X:

R: Key: 0x546DFEFA34…

RW: Key: 0x467D34EF83…

RX: Group Key: 0xA2D3498672…

Server 1: Domain 2

Client 1 Domain 1

$(\text{Read X})^{K}$ Key: 0x6647DBC9AC…

$(\text{data})^{K_{server}}$

Read Group

GACL

GACL verifier

Hash, Timestamp, Signature (group)

Group ACL:

Key: 0xA786EF889A…

Key: 0x6647DBC9AC…

Server 2: Domain 3

- Distributed Access Control List (ACL)
  - Contains list of attributes (Read, Write, Execute, etc) with attached identities (Here, we show public keys)
    - » ACLs signed by owner of file, only changeable by owner
    - » Group lists signed by group key
  - ACLs can be on different servers than data
    - » Signatures allow us to validate them
    - » ACLs could even be stored separately from verifiers

Analysis of Previous Scheme

- Positive Points:
  - Identities checked via signatures and public keys
    - » Client can't generate request for data unless they have private key to go with their public identity
    - » Server won't use ACLs not properly signed by owner of file
  - No problems with multiple domains, since identities designed to be cross-domain (public keys domain neutral)
- Revocation:
  - What if someone steals your private key?
    - » Need to walk through all ACLs with your key and change…!
    - » This is very expensive
  - Better to have unique string identifying you that people place into ACLs
    - » Then, ask Certificate Authority to give you a certificate matching unique string to your current public key
    - » Client Request: (request + unique ID)^{Cprivate}; give server certificate if they ask for it.
    - » Key compromise⇒must distribute "certificate revocation", since can't wait for previous certificate to expire.
  - What if you remove someone from ACL of a given file?
    - » If server caches old ACL, then person retains access!
    - » Here, cache inconsistency leads to security violations!

Analysis Continued

- Who signs the data?
  - Or: How does the client know they are getting valid data?
  - Signed by server?
    - » What if server compromised? Should client trust server?
  - Signed by owner of file?
    - » Better, but now only owner can update file!
    - » Pretty inconvenient!
  - Signed by group of servers that accepted latest update?
    - » If must have signatures from all servers ⇒ Safe, but one bad server can prevent update from happening
    - » Instead: ask for a threshold number of signatures
    - » Byzantine agreement can help here
- How do you know that data is up-to-date?
  - Valid signature only means data is valid older version
  - Freshness attack:
    - » Malicious server returns old data instead of recent data
    - » Problem with both ACLs and data
    - » E.g.: you just got a raise, but enemy breaks into a server and prevents payroll from seeing latest version of update
  - Hard problem
    - » Needs to be fixed by invalidating old copies or having a trusted group of servers (Byzantine Agrement?)

How fine-grained should access control be?

- Example of the problem:
  - Suppose you buy a copy of a new game from "Joe's Game World" and then run it.
  - It's running with your userid
    - » It removes all the files you own, including the project due the next day…
- How can you prevent this?
  - Have to run the program under *some* userid.
    - » Could create a second *games* userid for the user, which has no write privileges.
    - » Like the "nobody" userid in UNIX – can't do much
  - But what if the game needs to write out a file recording scores?
    - » Would need to give write privileges to one particular file (or directory) to your *games* userid.
  - But what about non-game programs you want to use, such as Quicken?
    - » Now you need to create your own private *quicken* userid, if you want to make sure tha the copy of Quicken you bought can't corrupt non-quicken-related files
  - But – how to get this right??? Pretty complex…

Authorization Continued

- Principle of least privilege: programs, users, and systems should get only enough privileges to perform their tasks
  - Very hard to do in practice
    - » How do you figure out what the minimum set of privileges is needed to run your programs?
  - People often run at higher privilege then necessary
    - » Such as the "administrator" privilege under windows
- One solution: Signed Software
  - Only use software from sources that you trust, thereby dealing with the problem by means of authentication
  - Fine for big, established firms such as Microsoft, since they can make their signing keys well known and people trust them
    - » Actually, not always fine: recently, one of Microsoft's signing keys was compromised, leading to malicious software that looked valid
  - What about new startups?
    - » Who "validates" them?
    - » How easy is it to fool them?

Involuntary Installation

- What about software loaded without your consent?
  - Macros attached to documents (such as Microsoft Word)
  - Active X controls (programs on web sites with potential access to whole machine)
  - Spyware included with normal products
- Active X controls can have access to the local machine
  - Install software/Launch programs
- Sony Spyware (October 2005)
  - About 50 recent CDs from Sony automatically install software when you played them on Windows machines
    - » Called XCP (Extended Copy Protection)
    - » Modify operating system to prevent more than 3 copies and to prevent uploading to itunes^{TM}
  - Side Effects:
    - » Reporting of private information to Sony
    - » Hiding of generic file names of form $sys_xxx; easy for other virus writers to exploit
    - » Hard to remove (crashes machine if not done carefully)
  - Vendors of virus protection software decide its spyware
    - » Computer Associates, Symantec, even Microsft

Enforcement

- Enforcer checks passwords, ACLs, etc
  - Makes sure the only authorized actions take place
  - Bugs in enforcer⇒things for malicious users to exploit
- In UNIX, superuser can do anything
  - Because of coarse-grained access control, lots of stuff has to run as superuser in order to work
  - If there is a bug in any one of these programs, you lose!
- Paradox
  - Bullet-proof enforcer
    - » Only known way is to make enforcer as small as possible
    - » Easier to make correct, but simple-minded protection model
  - Fancy protection
    - » Tries to adhere to principle of least privilege
    - » Really hard to get right
- Same argument for Java or C++: What do you make private vs public?
  - Hard to make sure that code is usable but only necessary modules are public
  - Pick something in middle? Get bugs and weak protection!

State of the World

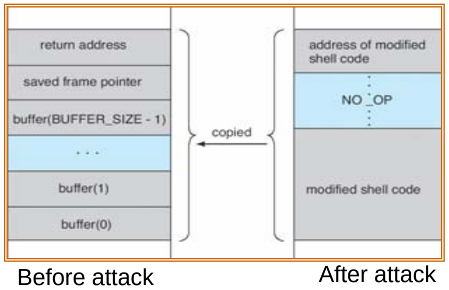
- State of the World in Security
  - Authentication: Encryption
    - » But almost no one encrypts or has public key identity
  - Authorization: Access Control
    - » But many systems only provide very coarse-grained access
    - » In UNIX, need to turn off protection to enable sharing
  - Enforcement: Kernel mode
    - » Hard to write a million line program without bugs
    - » Any bug is a potential security loophole!
- Some types of security problems
  - Abuse of privilege
    - » If the superuser is evil, we're all in trouble/can't do anything
    - » What if Kevin Mullaly (in charge of instructional resources) went crazy and deleted everybody's files (and backups)???
  - Imposter: Pretend to be someone else
    - » Example: in unix, can set up an .rhosts file to allow logins from one machine to another without retyping password
    - » Allows "rsh" command to do an operation on a remote node
    - » Result: send rsh request, pretending to be from trusted user→install .rhosts file granting

Other Security Problems

- Virus:
  - A piece of code that attaches itself to a program or file so it can spread from one computer to another, leaving infections as it travels
  - Most attached to executable files, so don't get activated until the file is actually executed
  - Once caught, can hide in boot tracks, other files, OS,
- Worm:
  - Similar to a virus, but capable of traveling on its own
  - Takes advantage of file or information transport features
  - Because it can replicate itself, your computer might send out hundreds or thousands of copies of itself
- Trojan Horse:
  - Named after huge wooden horse in Greek mythology given as gift to enemy; contained army inside
  - At first glance appears to be useful software but does damage once installed or run on your computer

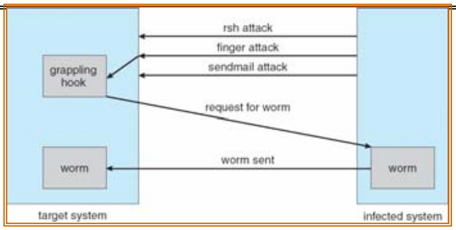
Security Problems: Buffer-overflow Condition

```
#define BUFFER SIZE 256
int process(int argc,
            char *argv[])
{
  char buffer[BUFFER SIZE];
  if (argc < 2)
       return -1;
  else {
       strcpy(buffer,argv[1]);
       return 0;
  }
}
```



- Technique exploited by many network attacks
  - Anytime input comes from network request and is not checked for size
  - Allows execution of code with same privileges as running program – but happens without any action from user!
- How to prevent?
  - Don't code this way! (ok, wishful thinking)
  - New mode bits in Intel, Amd, and Sun processors
    - » Put in page table; says "don't execute code in this page"

The Morris Internet Worm



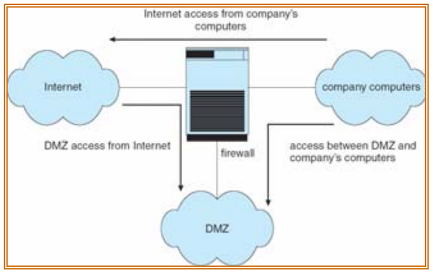
- Internet worm (Self-reproducing)
  - Author Robert Morris, a first-year Cornell grad student
  - Launched close of Workday on November 2, 1988
  - Within a few hours of release, it consumed resources to the point of bringing down infected machines
- Techniques
  - Exploited UNIX networking features (remote access)
  - Bugs in *finger* (buffer overflow) and *sendmail* programs (debug mode allowed remote login)
  - Dictionary lookup-based password cracking
  - Grappling hook program uploaded main worm program

Some other Attacks

- Trojan Horse Example: Fake Login
  - Construct a program that looks like normal login program
  - Gives "login:" and "password:" prompts
    - » You type information, it sends password to someone, then either logs you in or says "Permission Denied" and exits
  - In Windows, the "ctrl-alt-delete" sequence is supposed to be really hard to change, so you "know" that you are getting official login program
- Is SONY XCP a Trojan horse?
- Salami attack: Slicing things a little at a time
  - Steal or corrupt something a little bit at a time
  - E.g.: What happens to partial pennies from bank interest?
    - » Bank keeps them! Hacker re-programmed system so that partial pennies would go into his account.
    - » Doesn't seem like much, but if you are large bank can be millions of dollars
- Eavesdropping attack
  - Tap into network and see everything typed
  - Catch passwords, etc
  - Lesson: never use unencrypted communication!

Network Security Through Firewall

- How do I minimize the damage when security fails?
  - For instance: I make a mistake in the specification
  - Or: A bug lets something run that shouldn't?
- Firewall: Examines every packet to/from public internet
  - Can disable all traffic to/from certain ports
  - Can route certain traffic to DMZ (De-Militarized Zone)
    - » Semi-secure area separate from critical systems
  - Can do network address translation
    - » Inside network, computers have private IP addresses
    - » Connection from inside→outside is translated
    - » E.g. [10.0.0.2,port 2390] → [169.229.60.38,port 80]
[12.4.35.2,port 5592] → [169.229.60.38,port 80]


Internet access from company's computers
Internet
company computers
DMZ access from Internet
firewall
access between DMZ and company's computers
DMZ

Ken Thompson’s self-replicating program

- Bury Trojan horse in binaries, so no evidence in source
  - Replicates itself to every UNIX system in the world and even to new UNIX’s on new platforms. No visible sign.
  - Gave Ken Thompson ability to log into any UNIX system
- Two steps: Make it possible (easy); Hide it (tricky)
- Step 1: Modify login.c

```
A: if (name = “ken”)
       don’t check password
       log in as root
```

  - Easy to do but pretty blatant! Anyone looking will see.
- Step 2: Modify C compiler
  - Instead of putting code in login.c, put in compiler:

```
B: if see trigger1
       insert A into input stream
```

  - Whenever compiler sees trigger1 (say /*gobbledygook*/), puts A into input stream of compiler
  - Now, don’t need A in login.c, just need trigger1

Self Replicating Program Continued

- Step 3: Modify compiler source code:

```
C: if see trigger2
       insert B+C into input stream
```

  - Now compile this new C compiler to produce binary
- Step 4: Self-replicating code!
  - Simply remove statement C in compiler source code and place “trigger2” into source instead
    - » As long as existing C compiler is used to recompile the C compiler, the code will stay into the C compiler and will compile back door into login.c
    - » But no one can see this from source code!
- When porting to new machine/architecture, use existing C compiler to generate cross-compiler
  - Code will migrate to new architecture!
- Lesson: never underestimate the cleverness of computer hackers for hiding things!

Conclusion

- Distributed identity
  - Use cryptography (Public Key, Signed by PKI)
- Use of Public Key Encryption to get Session Key
  - Can send encrypted random values to server, now share secret with server
  - Used in SSL, for instance
- Authorization
  - Abstract table of users (or domains) vs permissions
  - Implemented either as access-control list or capability list
- Distributed ACL
  - Can include public keys or unique identifying strings
  - Sign all requests; server checks signature against ACL
- Issues with distributed storage example
  - Revocation: How to remove permissions from someone?
  - Integrity: How to know whether data is valid
  - Freshness: How to know whether data is recent
- Buffer-Overrun Attack: exploit bug to execute code