Operating system

Part XII: Protection [保护] & Security [安全]

By KONG LingBo (孔令波)

http://rjxy.bjtu.edu.cn:8080/teacherpages/lbkong/OS.2012/main.html

Security & Protection

- Security
  - Authentication [认证]
  - Authorization [授权]
- Protection
  - Protection matrix
  - Access control lists
  - Capabilities
  - Reference monitor
- Threats from the Internet
  - Trojan horses
  - Viruses

Definitions

- ***Security***: policy of authorizing accesses
  - Prevents intentional misuses of a system
- ***Protection***: the actual mechanisms implemented to enforce the specialized policy
  - Prevents either accidental or intentional misuses

For the safety of your computer system!

What kinds of intruders are there?

- Casual prying by nontechnical users
  - Curiosity
- Snooping by insiders
  - Often motivated by curiosity or money
- Determined attempt to make money
  - May not even be an insider
- Commercial or military espionage
  - This is very big business!

Snoop vi/n. 探听, 窥探; 管闲事

Accidents cause problems, too…

- Acts of God
  - Fires
  - Earthquakes
  - Wars (is this really an “act of God”?)
- Hardware or software error
  - CPU malfunction
  - Disk crash
  - Program bugs
- Human errors
  - Data entry
  - Wrong tape mounted
  - rm * .o

Security Goals

- ***Data confidentiality***:
  - secret data remains secret
- ***Data integrity***:
  - unauthorized users should not be able to modify data
- ***System availability***:
  - nobody can make a system unusable

Security Components

- ***Authentication***
  - determines who the user is
- ***Authorization***
  - determines who is allowed to do what
- ***Enforcement*** [强制检测]
  - makes it so people can do only what they are allowed to do

Sample Tools

- **Cryptography**
  - Can ensure confidentiality and integrity
  - Typically used for authentication
- **Firewalls**, passwords, **access control**
  - Authorization mechanisms
- Operating systems
  - Resource allocation
  - Monitoring and logging for audits
- Java bytecode verifier
  - Memory safety against malicious/defective code

We do not have adequate technology today!

Authentication [身份验证]

- The most common approach: ***passwords***
  - If I know the secret, the machine can assume that I'm the user
- Problems:
  1. Password storage
  2. Poor passwords

Password Storage

- ***Encryption***
  - Uses a key to transform the data
  - Difficult to reverse without the key
- UNIX stores encrypted passwords in **`/etc/passwd`**
  - Uses one-way transformations
  - Encrypts a typed password and compares encrypted passwords

Poor Passwords

- Short passwords
  - – Easy to crack
- Long passwords
  - – Tend to be written down somewhere

➔ Original UNIX
  - – Required only lower-case, 5-lettered passwords
  - – 26^5 or 1 million combinations
    - In 1975, it would take one day to crack one password
    - Today, we can go through all those combinations < 1 second

Partial Solutions

- Extend password with a unique number
- Require more complex passwords
  - 6 letters of upper, lower cases, numbers, and special characters
  - 70^6 or 100 billion combinations
  - Unfortunately, people st pick common words

Don’t forget your PSD then !

Authentication in Distributed Systems

- ***Private key*** encryption of data
  - **Encrypt**(Key, Plaintext) = Cipher text
  - Decrypt(Key, Cipher text) = Plaintext
- Hard to reverse without the key
  - With the plaintext and the cipher text, one cannot derive the key
- Provides secrecy and authentication, as long as the key stays secret

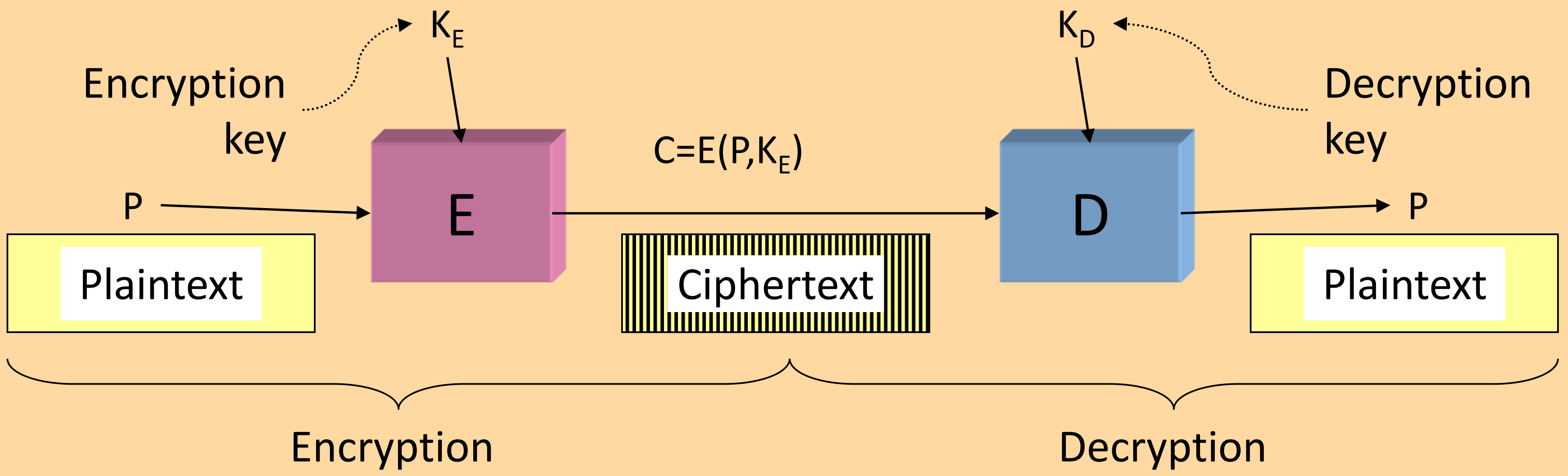
Cryptography [密码学]

- Goal: keep information from those who aren’t supposed to see it
  - Do this by “scrambling [混杂; 把…搅乱]” the data
- Use a well-known algorithm to scramble data
  - Algorithm has two inputs: data & key
  - Key is known only to “authorized” users
  - Relying upon the secrecy of the algorithm is a *very* bad idea (see WW2 Enigma for an example, WEP, WPA2 …)
- Cracking codes is ***very*** difficult, *Sneakers* and other movies notwithstanding

PPTs.2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

Cryptography basics

- Algorithms (E, D) are widely known
- Keys (K_E, K_D) may be less widely distributed
- For this to be effective, the ciphertext should be the only information that's available to the world
- Plaintext is known only to the people with the keys (in an ideal world…)



Encryption Algorithms

Symmetric

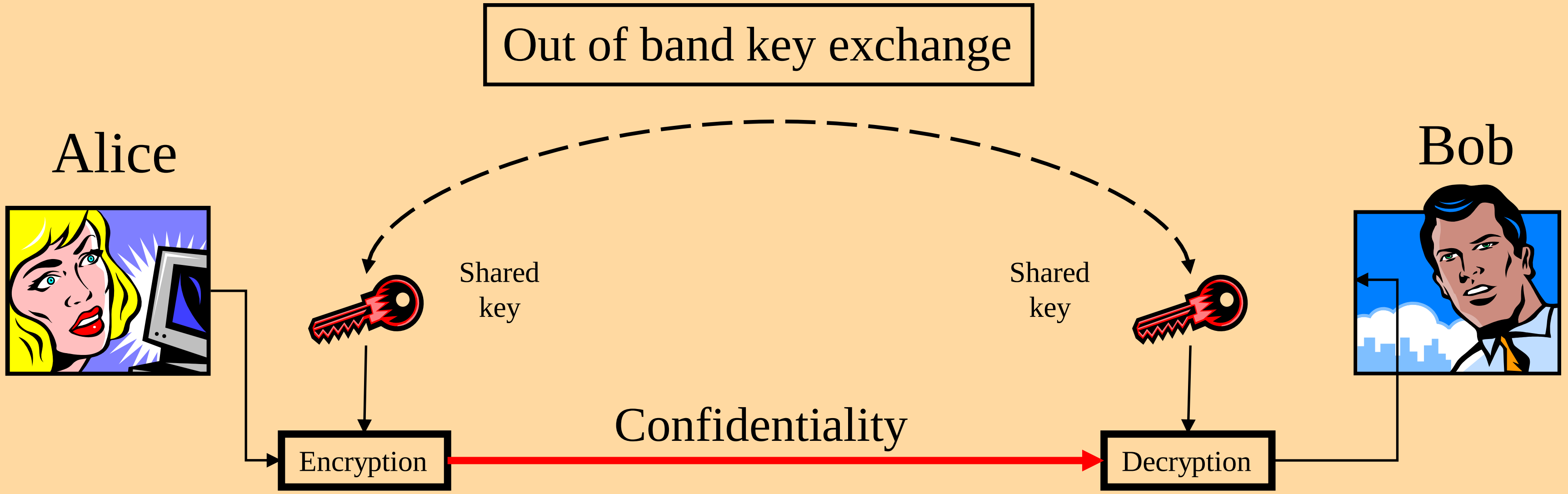
- Encryption and decryption use the same key
- Key must be secret (secret key)
- Best known: DES, AES, IDEA, Blowfish, RC5

Asymmetric

- Also known as Public Key Encryption
- Encryption and decryption keys *different*

DES – Data Encryption Standard, IDEA – International Data Encryption Algorithm, AES – Advanced Encryption System

Symmetric Encryption



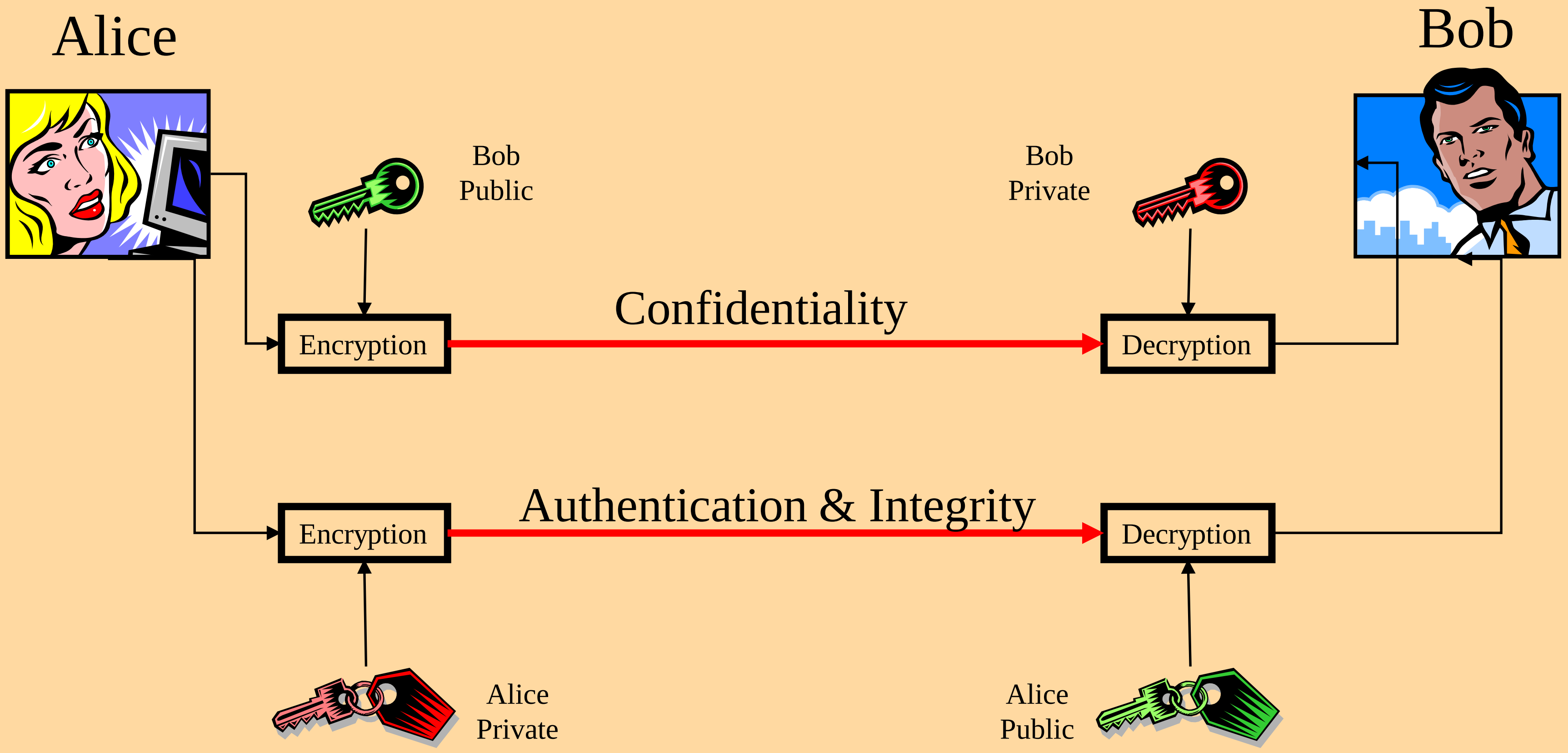
PPTs 2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

Asymmetric Encryption

- Two complementary keys
  - Private key (kept secret)
  - Public key (published)
- Private key is VERY difficult to compute from public key
- Encryption with one key can only be reversed with the other key
- Used in PGP (Pretty Good Privacy) &
  PKI (Public Key Infrastructure)
- Best known RSA & ECC, DSA for signatures

RSA Rivest Shami Adleman, ECC – Eliptic Curve Cryptography, DSA – Digital Signature Algorithm

Asymmetric Encryption (cont')



Modern encryption algorithms

- Data Encryption Standard (DES)
  - Uses 56-bit keys
  - Same key is used to encrypt & decrypt
  - Keys used to be difficult to guess
    - Needed to try 2^{55} different keys, on average
    - Modern computers can try millions of keys per second with special hardware
    - For $250K, EFF built a machine that broke DES quickly
- Current algorithms (AES, Blowfish) use 128 bit keys
  - Adding one bit to the key makes it twice as hard to guess
  - Must try 2^{127} keys, on average, to find the right one
  - At 10^{15} keys per second, this would require over 10^{21} seconds, or 1000 billion years!
  - Modern encryption isn’t usually broken by brute force…

Public Key Encryption

- Separates authentication from secrecy
- Involves a ***public key*** and ***private key***
  - Encrypt(Key$_{public}$, plaintext) = cipher text
  - Decrypt(Key$_{private}$, cipher text) = plaintext

  - Encrypt(Key$_{private}$, plaintext) = cipher text
  - Decrypt(Key$_{public}$, cipher text) = plaintext

PPTs.2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

Public Key Encryption

- Idea:
  - Private key is kept secret
  - Public key is advertised

➔ Encrypt(Key_{my_public}, "Hi, Andy")
  - Anyone can create it, but only I can read it (secrecy)

➔ Encrypt($Key_{my\_private}$, "I'm Andy")
  - Everyone can read it, but only I can create it (authentication)

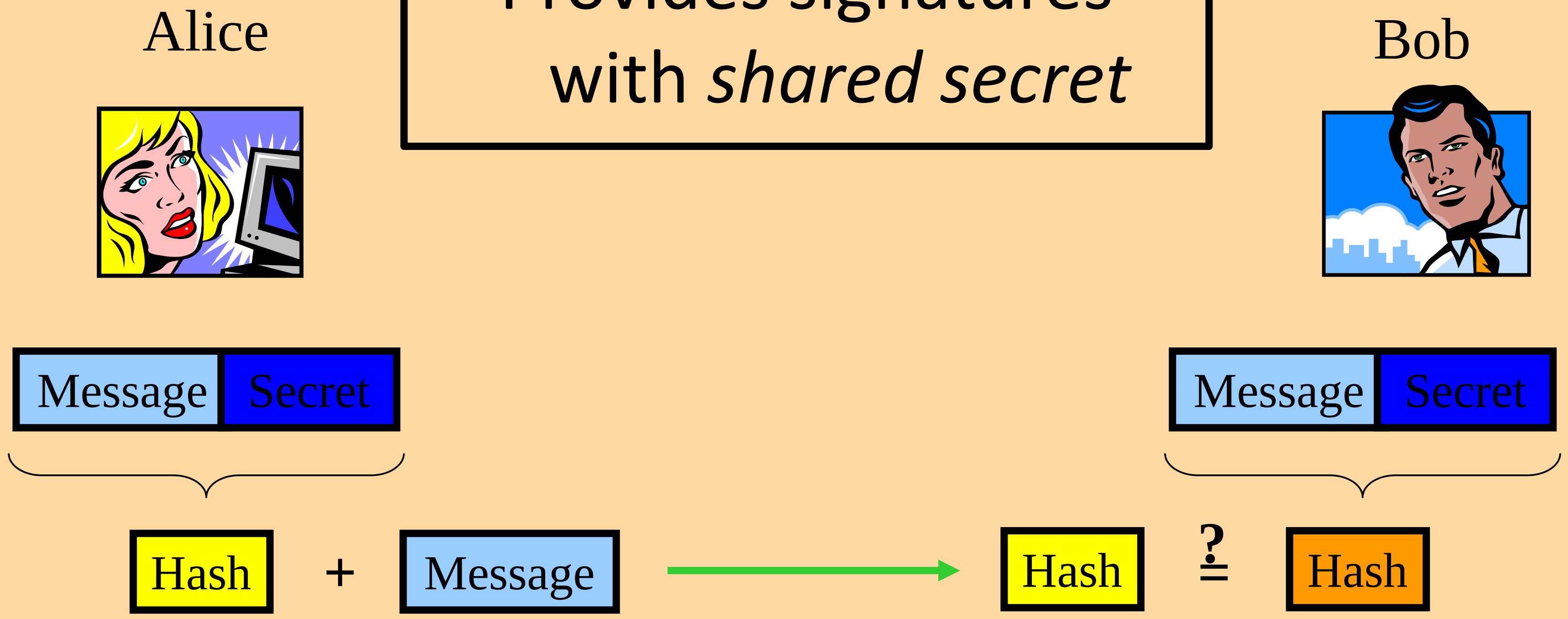
➔ Encrypt(Key_{your_public}, Encrypt($Key_{my\_private}$, "I know your secret"))
  - Only I can create it, and only you can read it

Digital Signatures with Public Keys

- Suppose K is public key and k is private key for Alice, and encryption/decryption is commutative [交换的; 代替的]:
  D(E(M,K), k) = E(D(M,k),K)=M
- To sign a message M, Alice simply sends D(M,k)
- Receiver uses Alice’s public key to compute E(D(M,k),K), to retrieve M
  – Authenticity of signature because only Alice knows the private key k
- The scheme is made more efficient by computing D(H(M),k), where H(M) is the *secure hash* of M
  – Hashing gives a constant size output
  – Hard to invert

Hash Functions cont’d

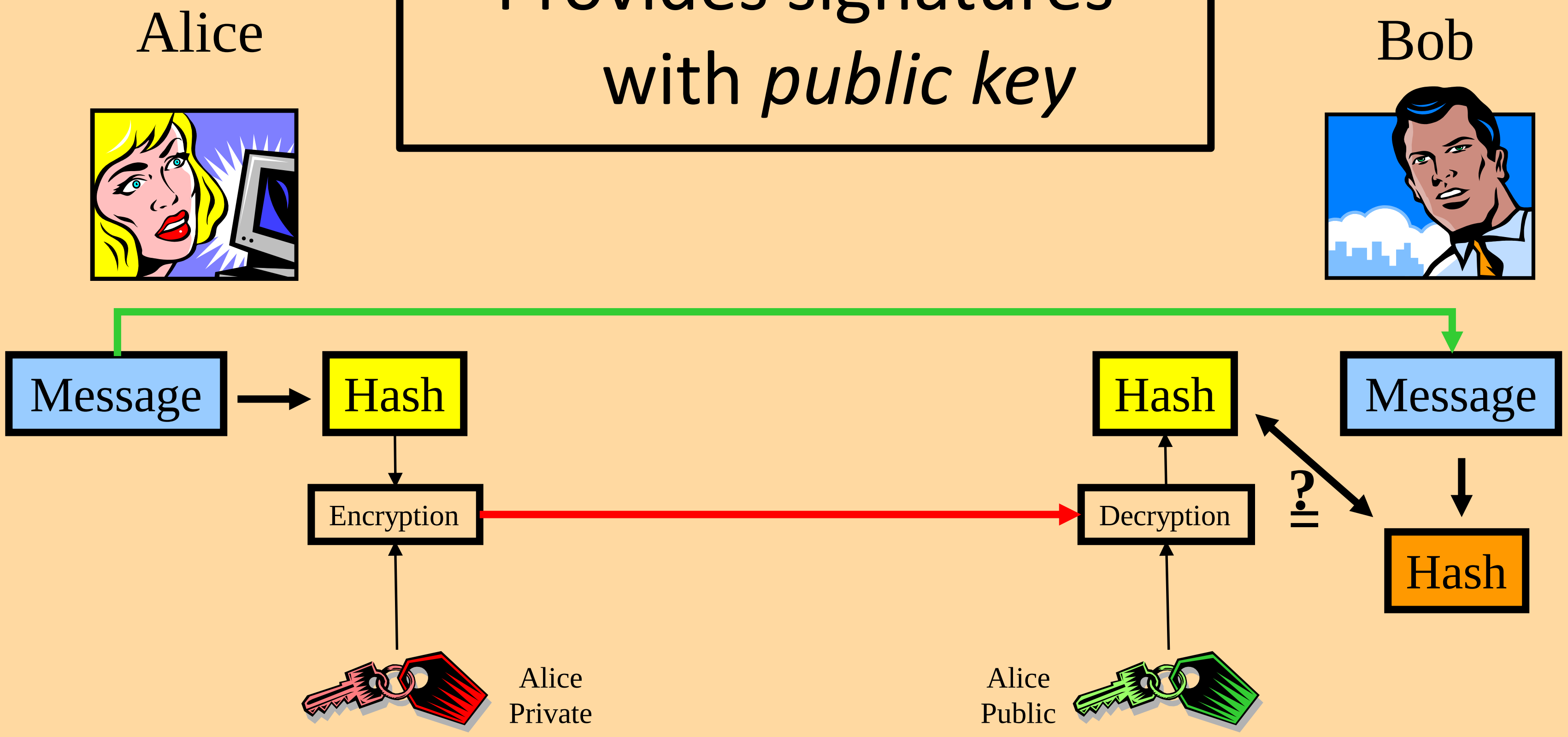
Provides signatures with *shared secret*



Hash Functions cont’d

PPTs 2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

Provides signatures with *public key*



Security & Protection

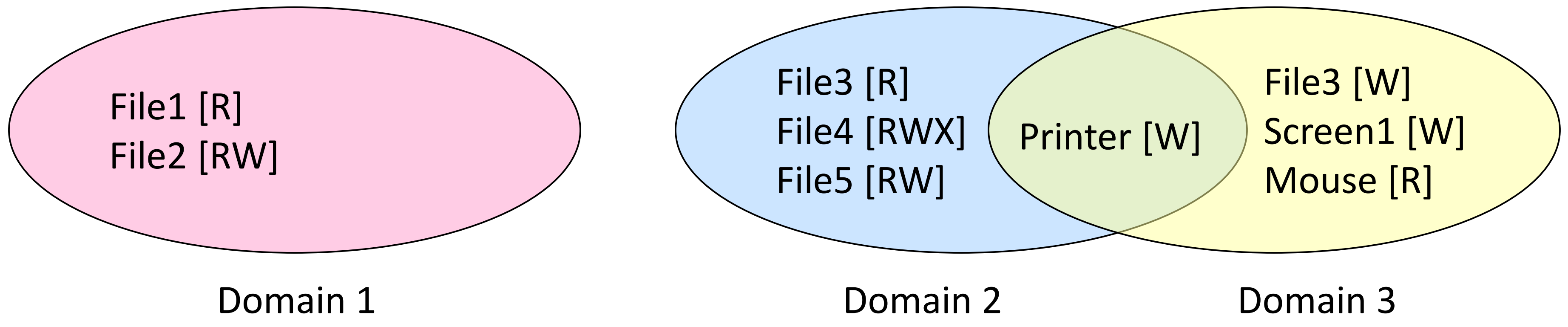
- Security
  - Authentication
  - Authorization
- Protection
  - Protection matrix
  - Access control lists
  - Capabilities
  - Reference monitor
- Threats from the Internet
  - Trojan horses
  - Viruses

Protection

- Security is mostly about *mechanism*
  - How to enforce policies
  - Policies largely independent of mechanism
- Protection is about specifying policies
  - How to decide who can access what?
- Specifications must be
  - Correct
  - Efficient
  - Easy to use (or nobody will use them!)

Protection domains

- Three protection domains
  - Each lists objects with permitted operations
- Domains can share objects & permissions
  - Objects can have different permissions in different domains
  - There need be no overlap between object permissions in different domains
- How can this arrangement be specified more formally?



Protection matrix

Domain	File1	File2	File3	File4	File5	Printer1	Mouse
1	Read	Read Write					
2			Read	Read Write Execute	Read Write	Write	
3			Write			Write	Read

- Each domain has a row in the matrix
- Each object has a column in the matrix
- Entry for <object,column> has the permissions
- Who’s allowed to modify the protection matrix?
  - What changes can they make?
- How is this implemented efficiently?

Domains as objects in the protection matrix

Domain	File1	File2	File3	File4	File5	Printer1	Mouse	Dom1	Dom2	Dom3
1	Read	Read Write						Modify		
2			Read	Read Write Execute	Read Write	Write		Modify		
3			Write			Write	Read		Enter	

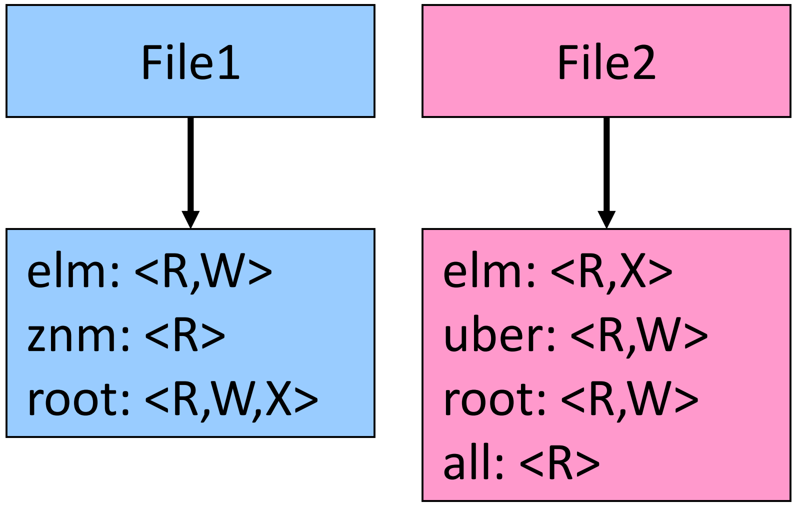
- Specify permitted operations on domains in the matrix
  - Domains may (or may not) be able to modify themselves
  - Domains can modify other domains
  - Some domain transfers permitted, others not
- Doing this allows flexibility in specifying domain permissions
  - Retains ability to restrict modification of domain policies

Representing the protection matrix

- Need to find an efficient representation of the protection matrix (also called the *access matrix*)
- Most entries in the matrix are empty!
- Compress the matrix by:
  – Associating permissions with each object: *access control list*
  – Associating permissions with each domain: *capabilities*
- How is this done, and what are the tradeoffs?

Access control lists

- Each object has a list attached to it
- List has
  - Protection domain
    - User name
    - Group of users
    - Other
  - Access rights
    - Read
    - Write
    - Execute (?)
    - Others?
- No entry for domain => no rights for that domain
- Operating system checks permissions when access is needed

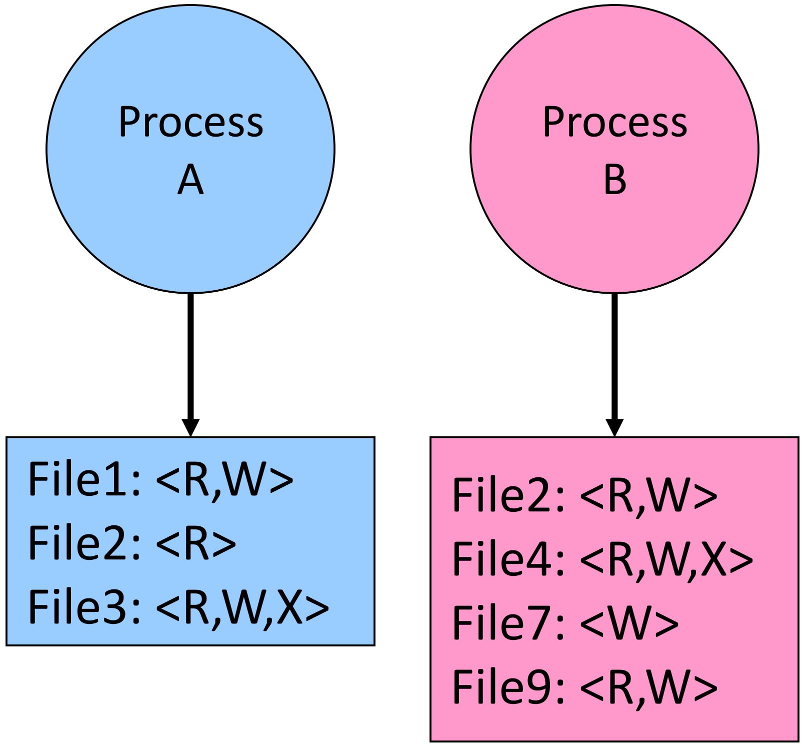


Access control lists in the real world

- Unix file system
  - Access list for each file has exactly three domains on it
    - User (owner)
    - Group
    - Others
  - Rights include read, write, execute: interpreted differently for directories and files
- AFS
  - Access lists only apply to directories: files inherit rights from the directory they're in
  - Access list may have many entries on it with possible rights:
    - read, write, lock (for files in the directory)
    - lookup, insert, delete (for the directories themselves),
    - administer (ability to add or remove rights from the ACL)

Capabilities

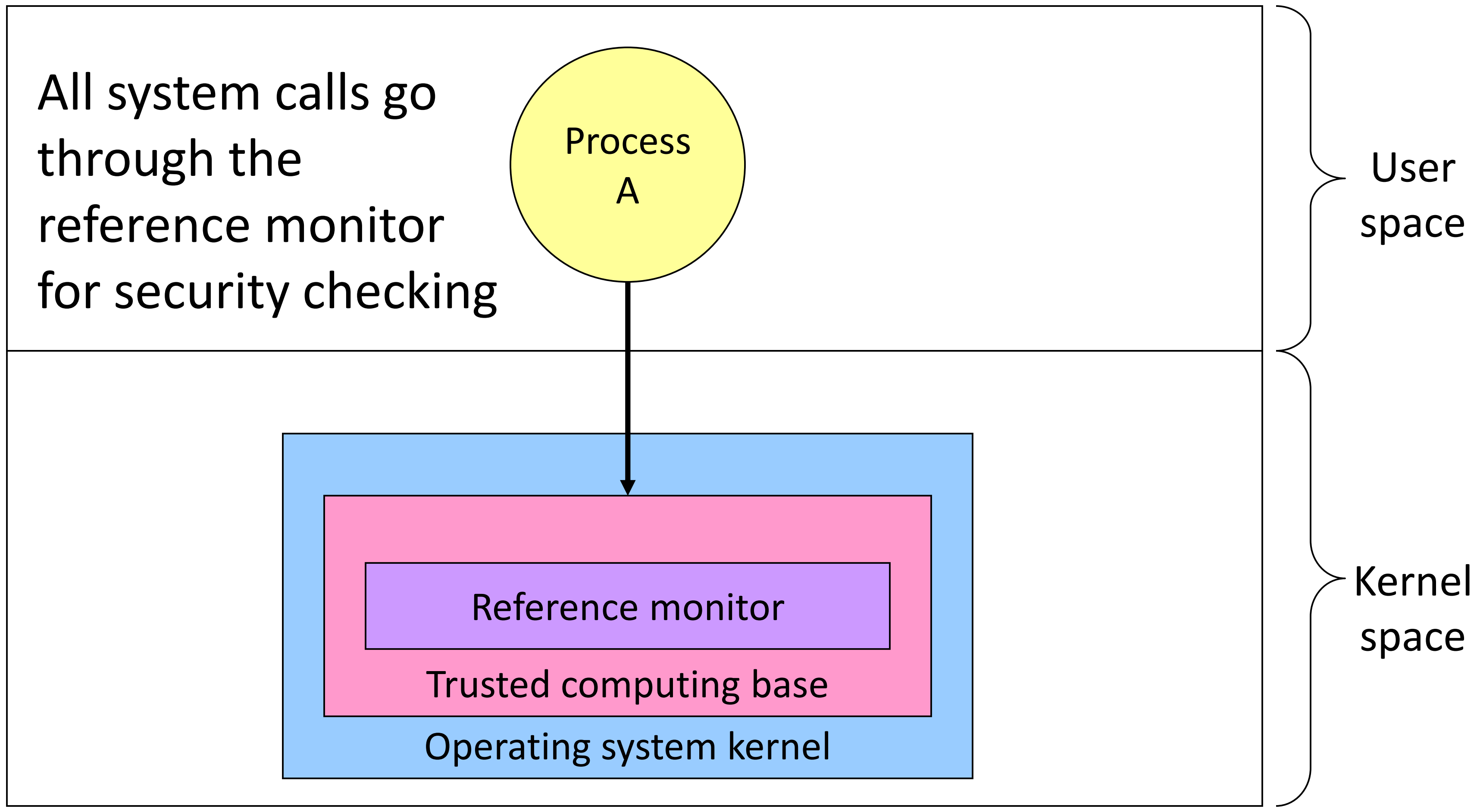
- Each process has a capability list
- List has one entry per object the process can access
  - Object name
  - Object permissions
- Objects not listed are not accessible
- How are these secured?
  - Kept in kernel
  - Cryptographically secured



Protecting the access matrix: summary

- OS must ensure that the access matrix isn't modified (or even accessed) in an unauthorized way
- Access control lists
  - Reading or modifying the ACL is a system call
  - OS makes sure the desired operation is allowed
- Capability lists
  - Can be handled the same way as ACLs: reading and modification done by OS
  - Can be handed to processes and verified cryptographically later on
  - May be better for widely distributed systems where capabilities can't be centrally checked

Reference monitor



Security & Protection

- Security
  - Authentication
  - Authorization
- Protection
  - Protection matrix
  - Access control lists
  - Capabilities
  - Reference monitor
- Threats from the Internet
  - Trojan horses
  - Viruses

Network Security

- External threat
  - code transmitted to target machine
  - code executed there, doing damage
- Goals of virus writer
  - quickly spreading virus
  - difficult to detect
  - hard to get rid of
- Virus = program can reproduce itself
  - by attaching its code to another program
  - additionally, do harm
- Worm
  - self-replicating

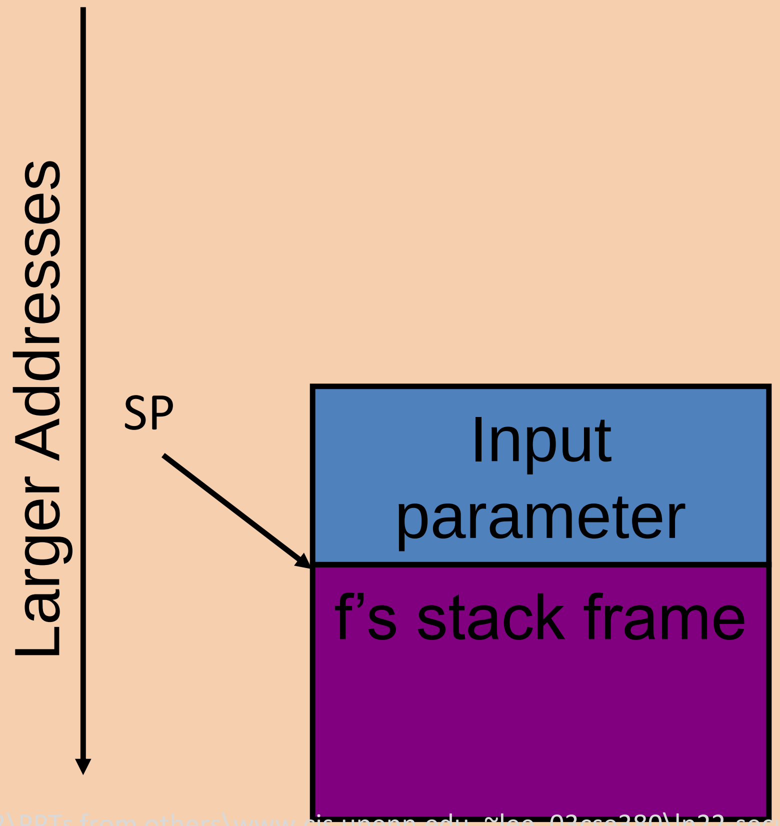
Buffer Overflow Attacks

- > 50% of security incidents reported at CERT (see cert.org) are due to buffer overflow attacks
- C and C++ programming languages don’t do array bounds checks
  - In particular, widely used library functions such as strcpy, gets
- Exploited in many famous attacks (read your Windows Service Pack notes)

PPTs.2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

C’s Control Stack

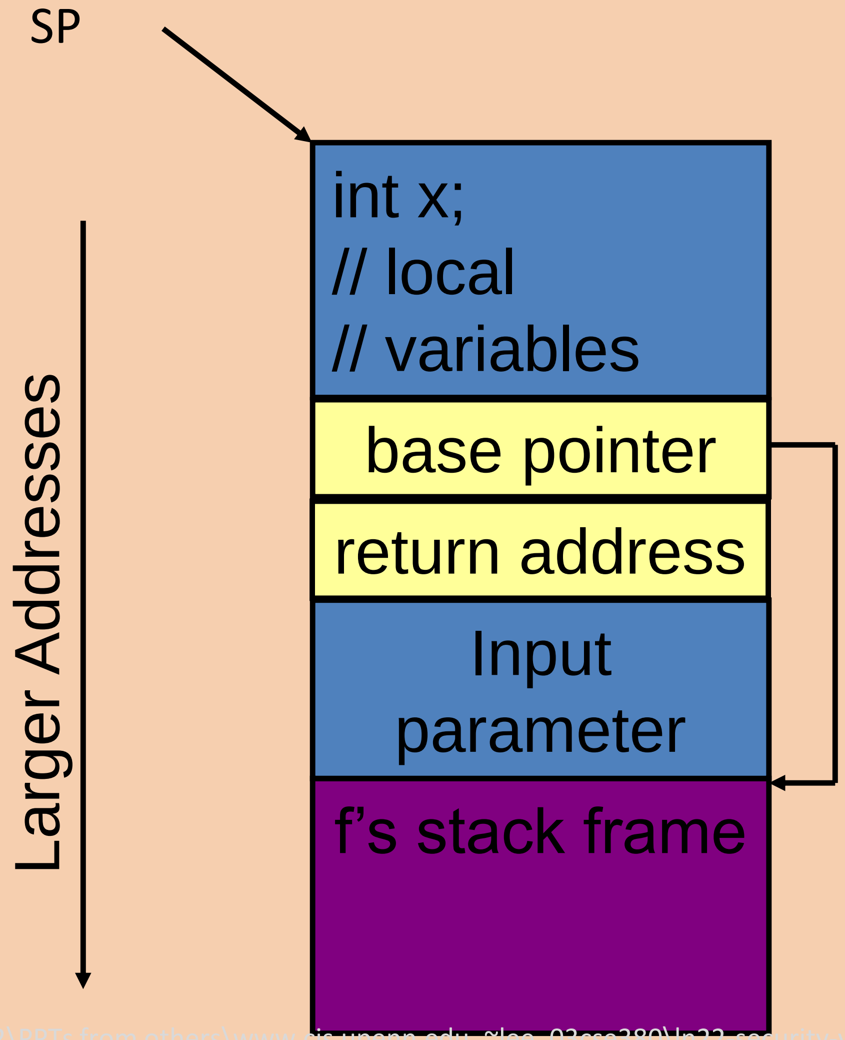
```
f() {
  g(parameter);
}

g(char *args) {
  int x;
  // more local
  // variables
  ...
}
```



Before calling g

C’s Control Stack

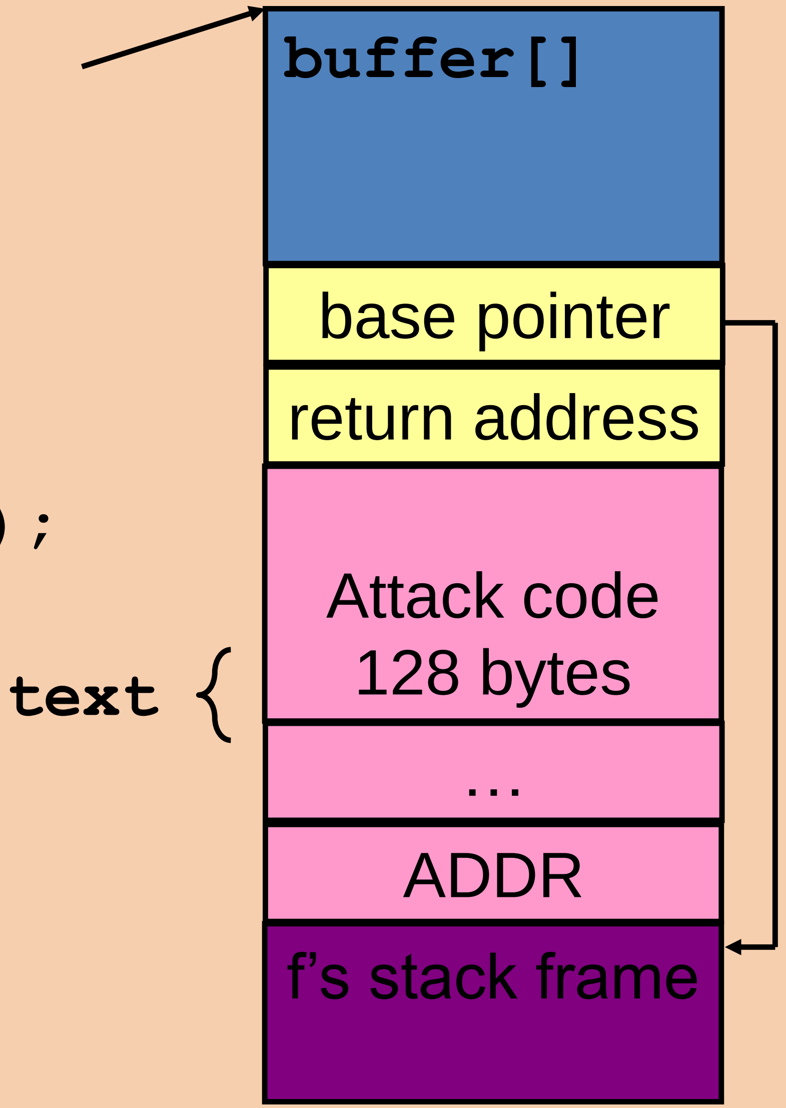
```
f() {
  g(parameter);
}

g(char *args) {
  int x;
  // more local
  // variables
  ...
}
```



After calling g

Buffer Overflow Example

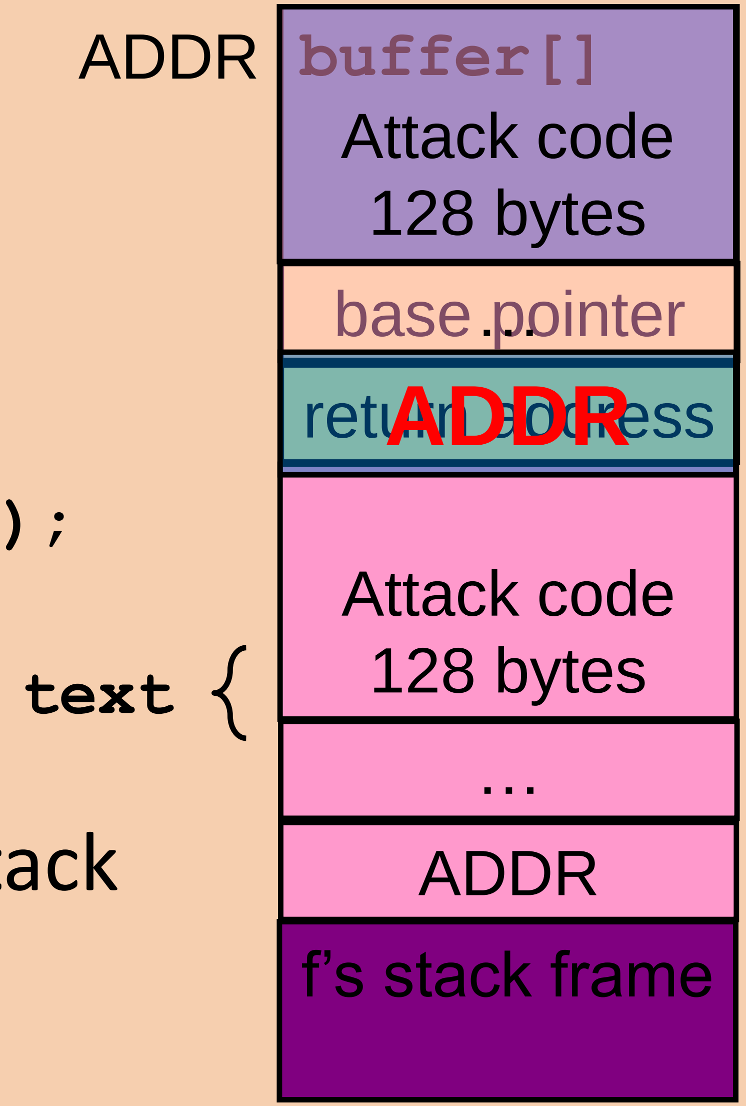
```
g(char *text) {
  char buffer[128];
  strcpy(buffer, text);
}
```



PPTs.2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

Buffer Overflow Example

```
g(char *text) {
  char buffer[128];
  strcpy(buffer, text);
}
```



Upon return from g, attack code gets executed !

Solutions

- Don’t write code in C
  - Use a safe language instead (Java, C#, …)
  - Not always possible (low level programming)
  - Doesn’t solve legacy code problem
- Link C code against safe version of libc
  - May degrade performance unacceptably
- Software fault isolation
  - Instrument executable code to insert checks
- Program analysis techniques
  - Examine program to see whether “tainted” data is used as argument to strcpy

Trojan horses [特洛伊木马]

- Free program made available to unsuspecting user
  - Actually contains code to do harm
  - May do something useful as well…
- Altered version of utility program on victim's computer
  - Trick user into running that program
- Example (getting superuser access on CATS?)
  - Place a file called **ls** in your home directory
    - File creates a shell in /tmp with privileges of whoever ran it
    - File then actually runs the real ls
  - Complain to your sysadmin that you can’t see any files in your directory
  - Sysadmin runs ls in your directory
    - Hopefully, he runs *your* ls rather than the real one (depends on his search path)

Virus damage scenarios

- Blackmail [敲诈，勒索，讹诈]
- Denial of service as long as virus runs
- Permanently damage hardware
- Target a competitor's computer
  - Do harm
  - Espionage
- Intra-corporate dirty tricks
  - Practical joke
  - Sabotage [阴谋破坏，蓄意破坏] another corporate officer's files

How viruses work

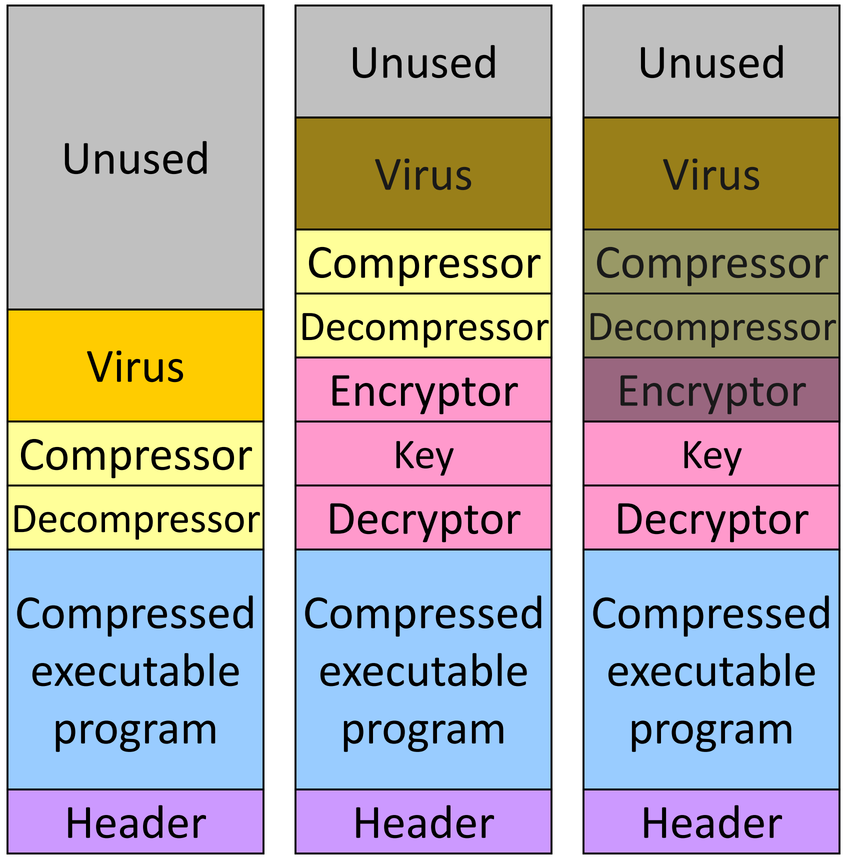
- Virus language
  - Assembly language: infects programs
  - “Macro” language: infects email and other documents
    - Runs when email reader / browser program opens message
    - Program “runs” virus (as message attachment) automatically
- Inserted into another program
  - Use tool called a “dropper”
  - May also infect system code (boot block, etc.)
- Virus dormant until program executed
  - Then infects other programs
  - Eventually executes its “payload”

Worms vs. viruses

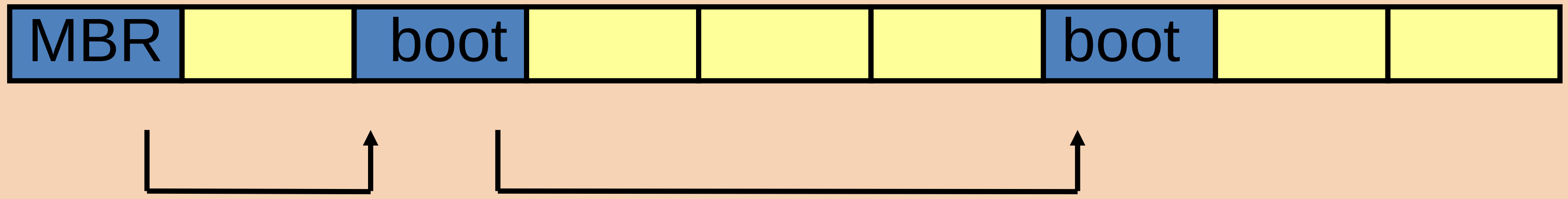
- Viruses require other programs to run
- Worms are self-running (separate process)
- The 1988 Internet Worm
  - Consisted of two programs
    - Bootstrap to upload worm
    - The worm itself
  - Exploited bugs in sendmail and finger
  - Worm first hid its existence
  - Next replicated itself on new machines
  - Brought the Internet (1988 version) to a screeching halt

Using encryption to hide a virus

- Hide virus by encrypting it
  - Vary the key in each file
  - Virus “code” varies in each infected file
  - Problem: lots of common code still in the clear
    - Compress / decompress
    - Encrypt / decrypt
- Even better: leave only decryptor and key in the clear
  - Less constant per virus
  - Use polymorphic code (more in a bit) to hide even this

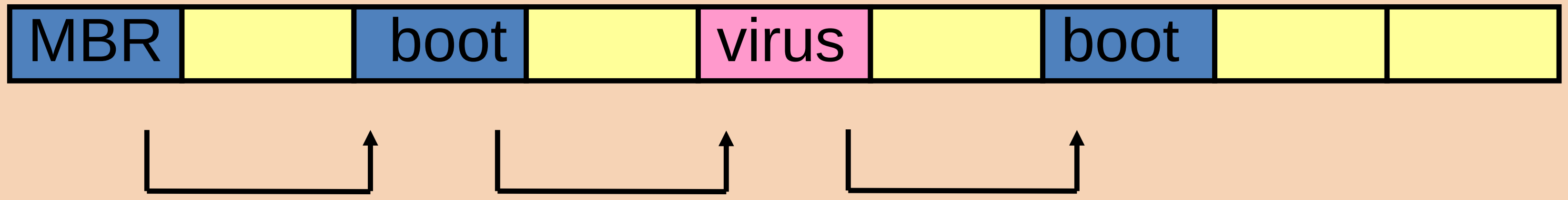


Bootstrap Viruses



- Bootstrap Process:
  - Firmware (ROM) copies MBR (master boot record) to memory, jumps to that program
- MBR (or Boot Sector)
  - Fixed position on disk
  - “Chained” boot sectors permit longer Bootstrap Loaders

Bootstrap Viruses



- Virus breaks the chain
- Inserts virus code
- Reconnects chain afterwards

Why the Boot Sector?

- Automatically executed *before* OS is running
  - Also before detection tools are running
- OS hides boot sector information from users
  - Hard to discover that the virus is there
  - Harder to fix

- Any good virus scanning software scans the boot sectors

PPTs.2012\PPTs from others\www.cis.upenn.edu_~lee_03cse380\ln22-security-v3.ppt

The Internet Worm

- In 1988, a Cornell graduate student, RTM (Robert Tappan Morris Jr.), released a worm into the Internet
- The worm used three attacks
  - **`Rsh (a kind of shell)`**
  - **`fingerd`**
  - **`Sendmail`**
- Some machines trust other machines, the use of **`rsh`** was sufficient to get into a remote machine without authentication

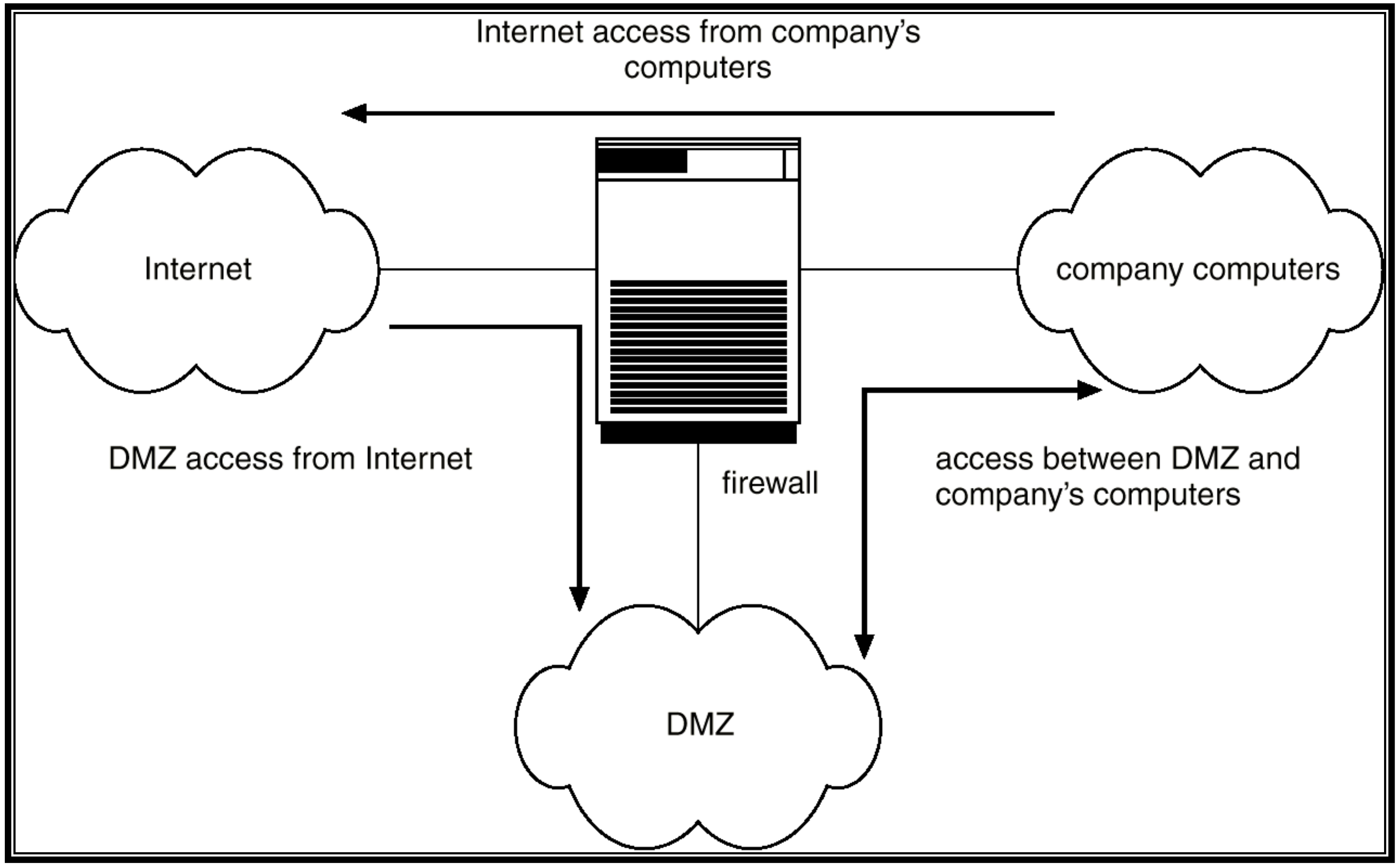
The Internet Worm (cont’)

- **`finger`** command did not check the input buffer size
  - **`finger name@location`**
  - Overflow the buffer
  - Overwrite the return address of a procedure
  - Jump and execute a shell (under root privilege)
- **`sendmail`** allowed the worm to mail a copy of the code and get it executed
- The worm was caught due to multiple infections
  - People noticed the high CPU load

FireWall [防火墙]

- A firewall is a computer or router placed between trusted and untrusted hosts.
- The firewall limits network access between these two security domains.
  - It can also limit connections based on source or destination address or direction of the connection
- DMZ (demilitarized [非武装的] zone) is a semi-secure and semi-trusted network
  - The Internet can connect to the DMZ but not directly to the company’s computers. The company’s computers connect to the DMZ and the firewall protects the company’s computers from illegal access from the Internet.

Network Security Through Domain Separation Via Firewall


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

Intrusion Detection [入侵检测]

- Detect attempts to intrude into computer systems
  - Signature-based detection: specific behaviors patterns (signatures) of input or network traffic
    - Multiple failed logon attempts to an account
    - Only detects known attacks based on signature not new ones
  - Anamoly-based detection
    - Unusual logon time for a user
    - Detect possible use of buffer overflow
    - Difficulty – can detect new attacks but how to you decide what is “normal”

Intrusion Detection (cont’)

- Detection methods:
  - Audit trail processing – match security relevant events against attack signatures or analyzed for anomalous behavior
  - Tripwire [绊网；引爆线；警报拉发线] file system integrity checking tool
    - UNIX software that checks if certain files and directories have been altered – i.e. password files
    - System call monitoring – monitors process system calls to detect in real-time when a process is deviating from its expected system-call behavior
      - Can be used to detect buffer overflow exploitation

Windows NT Example

- Security is based on user accounts where each user has a security ID.
- Uses a subject model to ensure access security. A
  - subject tracks and manages permissions for each program that a user runs.
- Each object in Windows NT has a security attribute defined by a security descriptor.
  - For example, a file has a security descriptor that indicates the access permissions for all users.

Security in Java

- Java is a type safe language
  - Compiler rejects attempts to misuse variable
- No “real” pointers
  - Can’t simply create a pointer and dereference it as in C
- Checks include …
  - Attempts to forge pointers
  - Violation of access restrictions on private class members
  - Misuse of variables by type
  - Generation of stack over/underflows
  - Illegal conversion of variables to another type
- Applets can have specific operations restricted
  - Example: don’t allow untrusted code access to the whole file system