Operating System Security, Con't

CS 239, Spring 1996

June 3, 1996

Outline

- Single system security
  - Memory, files, processes, devices
  - Dealing with intruders
  - Malicious programs
- Distributed system security
  - Using encryption
  - Secure distributed applications

Single System Security

Only worrying about the security of a single machine (possibly a multiprocessor)

One operating system is in control

Threats come from multiple users

- Or from external access

Protecting Memory

Virtual memory offers strong protection tools

- Model prevents naming another user's memory

What about shared memory?

- Use access control mechanisms
- Backed up by hardware protection on pages

Protecting Files

Unlike memory, files are in a shared namespace

Requires more use of access controls

Typically, access checked on open

- System assumes user has right to continue using open file

File Access Control in Unix

Every file has an owning user and group

Access permissions settable for read, write, and execute

For owning user, owning group, everyone else

Processes belong to one user

- And possibly multiple groups

Files opened for particular kinds of access

Protecting Processes

Most of a process' state not addressable externally

But IPC channels allow information to flow

So security must be applied at IPC points

Protecting IPC

Typically, IPC requires cooperation from both ends

So a major question is authentication

- Does this channel connect where I think it does?

OS guarantees identity, ownership of other process

Limiting IPC Access

Each party to IPC has control over what is done on his side

Some IPC mechanisms allow differing modes of access for different users

So access control required for such cases

Protecting Devices

Generally treated similarly to files

But special care is necessary

- In some cases, a mistake allows an intruder unlimited access
- E.g., if you let him write any block on a disk drive

Controlling IPC Access in Windows NT

General model related to file access control

Processes try to access objects

- Objects include IPC entities

On first access, request desired access rights

- Set of granted access rights returned

System checks granted access rights on each attempted access

Beware of Back Doors

Many systems provide low-level ways to access various resources

- `/dev/kmem`
- raw devices
- pipes stored in the file system

The lock on the back door must be as strong as the lock on the front door

Intruders

Modern systems usually allow remote access

- From terminals
- From modems
- From the network

Intruders can use all of these to break in

How Intruders Get in

Usually by masquerading as a legitimate user

Less frequently by inserting commands through insecure entry points

- `finger` daemons
- holes in electronic mail
- making use of interpreters that access data remotely

Detecting Intruders

The sooner detected, the better

Systems that detect and eject intruders quickly are less attractive targets

Information gained from detecting intruders can be used to prevent further intrusions

Detection presumes you can differentiate the behavior of authorized users and intruders

Some Approaches to Detecting Intruders

- Statistical anomaly detection
  Based on either -
    Overall system activity
    Individual user profiles
- Rule-based detection
  Rules that detect anomalies
  Penetration expert systems

Audit Records

Keep track of everything done on system

Powerful tool for detecting intruders

Used to build detection mechanisms

Can use either general accounting info or specially gathered data

Also invaluable if you decide to prosecute

Must be carefully protected to be valuable

Malicious Programs

Clever programmers can get software to do their dirty work for them

Programs have several advantages for these purposes

- Speed
- Mutability
- Anonymity

Kinds of Malicious Programs

- Trojan horses
- Trapdoors
- Logic bombs
- Worms
- Viruses

Trojan Horses

Seemingly useful program that contains code that does harmful things

Unsuspecting users run the Trojan horse to get the advertised benefit

At which time the Greeks spring out and slaughter your system

Particularly dangerous in compilers

Trapdoors

A secret entry point into an otherwise legitimate program

Typically inserted by the writer of the program

Most often found in login programs or programs that use the network

But also found in system utilities

Logic Bombs

Like trapdoors, typically in a legitimate program

A piece of code that, under certain conditions, "explodes"

Also like trapdoors, typically inserted by program authors

Worms

Programs that seek to move from system to system

Making use of various vulnerabilities

Other malicious behavior can also be built in

The Internet worm is the most famous example

Can spread very, very rapidly

Viruses

A program that can infect other programs

Infected programs in turn infect others

Along with mere infection, Trojan horses, trapdoors, or logic bombs can be included

Like worms, viruses can spread very rapidly

How Do Viruses Work?

When a program is run, it typically has the full privileges of its running user

Including write privileges for some other programs

A virus can use those privileges to replace those programs with infected versions

Typical Virus Actions

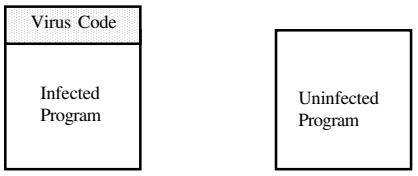
1). Find uninfected writable programs

2). Modify those programs

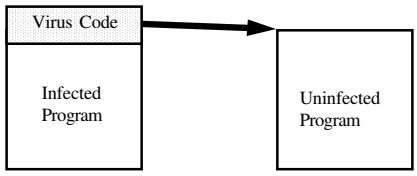
3). Perform normal actions of infected program

4). Do whatever other damage is desired

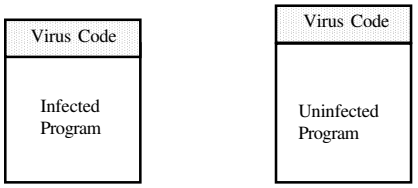
Before the Infected Program Runs



The Infected Program Runs



Infecting the Other Program



How Do Viruses Fit Into Programs?

- Prepended
- Postpended
- Copy program and replace
- Cleverly fit into the cracks

Some viruses take other measures to hide modifications

Dealing With Viruses

- Prevention of infection
- Detection and eradication
- Containment

Preventing the Spread of Viruses

Don't import untrusted programs

But who can you trust?

Viruses have been found in commercial shrink-wrap software

Trusting someone means not just trusting their honesty, but also their caution

Other Prevention Measures

- Scan incoming programs for viruses

  Some viruses are designed to hide
- Limit the targets viruses can reach
- Monitor updates to executables carefully

  Requires a broad definition of "executable"

Virus Detection

Many viruses have detectable signatures

But some work hard to hide them

Smart scanners can examine programs for virus-like behavior

Checksums attached to programs can detect modification

If virus smart enough to generate checksum itself, digitally sign it

Virus Eradication

Tedious, because you must be thorough

Restore clean versions of everything

Take great care with future restoration of backups

Containment

Run suspect programs in an encapsulated environment

Limiting their forms of access to prevent virus spread

Requires versatile security model and strong protection guarantees

Security in Distributed Systems

A substantially harder problem

Many single-system mechanisms are based on trusting a central operating system

Single-system mechanisms often assume secure communications channels

Single-system mechanisms can (in principle) have access to all relevant data

Security Mechanisms for Distributed Systems

- Encryption
- Authentication
- Firewalls

Encryption for Distributed Systems

Can protect secrecy of data while on insecure links

Can also prevent modification and many forms of fabrication attacks

But keys are a tricky issue

Encryption Keys and Distributed System Security

To gain benefit from encryption, communicating entities must share a key

Each separate set of entities need a different key

How do you securely distribute keys?

Problems of Key Distribution

- Key must be kept secret
- Key must be generated by trusted authority
- Must be sure key matches intended use
- Must be sure keys aren't reused
- Must be quick and automatic

Key Distribution Schemes

- Manual distribution by one party
- Manual distribution by third party
- Use existing key to send new key
- Key servers

Key Servers

Trusted third party that can provide good keys on demand

Must be a program, for full utility

Typically on a separate machine

Tremendous care must be taken to ensure secure communications with the key server

Authentication For Distributed Systems

When a message comes in over the net, how do you tell who sent it?

Generally with some form of *digital signature*

Must be unique to signing user

And also unique to the message

Digital Signatures

A *digital signature* is a guarantee that an electronic document was created by a particular individual

Basic mechanism for authentication

Vital for electronic commerce, secure electronic mail, etc.

S = signature(M)

Desirable Properties of Digital Signatures

- Easy to generate & verify
- Non-forgeable
- Unique (or close to)
- Non-repudiable
- Storable

Providing Digital Signatures

Encryption with a secret key has some of these properties

- Encrypt entire message
- Check signature by decrypting

$\mathbf{S = E(M,K_e)}$

But normal encryption has problems for digital signatures

Problems of Using Encryption for Digital Signatures

- Both parties can create same message
  - With same signature
- One key per pair of users required
- Signature is as large of message
  - Poor storage properties
- Hard to handle multiple signatures per message

Public Key Cryptography

An asymmetrical form of cryptography

Different keys for encrypt/decrypt

$$C = P(M, K_e)$$

$$M = P(C, K_d)$$

Important insight - only one key need be secret!

The Public Key

Everyone has one key that is known by all - it's not a secret

For user X,

$K_{d,X}$

This key can decrypt any message the owner sends

Thus, if it decrypts with the public key, it must be from the owner

The Private Key

The secret part of the key pair

Only known to owner X

$K_{e,X}$

Used to encrypt all messages that require authentication

Public Key Cryptography For Authentication

Originator X of message encrypts it by

$$C = P(M, K_{e,X})$$

Receiver Y decrypts by

$$M = P(C, K_{d,Y})$$

Public Key Cryptography For Encryption

Good public key schemes allow encryption with the public key

Must know the private key to decrypt

So, to send an encrypted message to Y,

$$C = P(M, K_{d,Y})$$

To decrypt it,

$$M = P(M, K_{e,Y})$$

Public Key Cryptography and Digital Signatures

User X wants to sign a message M sent to user Y

Calculate a characteristic Z of message M (checksum or something similar)

$$S = P(Z, K_{e,X})$$

Send both M and S to Y

Checking a Public Key Digital Signature

Y calculates the characteristic $\mathbf{Z_M}$ of **M**

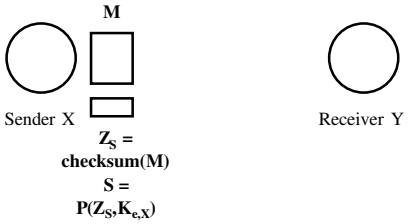
Then Y checks the signature

$\mathbf{Z_S = P(S,K_{d,X})}$

If $\mathbf{Z_M == Z_S}$, the signature is valid

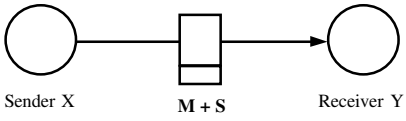
Public Key Digital Signature Diagram, Step 1

Sign the message



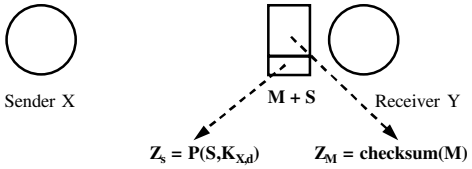
Public Key Digital Signature Diagram, Step 2

Send the message plus the signature



Public Key Digital Signature Diagram, Step 3

Check the signature



If $Z_S == Z_M$, the signature is valid

How Does This Scheme Handle Various Attacks?

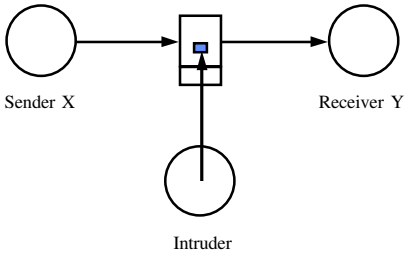
What if an intruder changes the message?

What if someone replays a message?

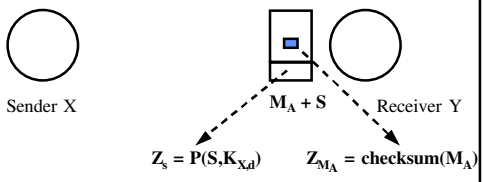
What if the sender denies a message he sent?

What if the receiver tries to alter the message?

Intruder Alteration Diagram

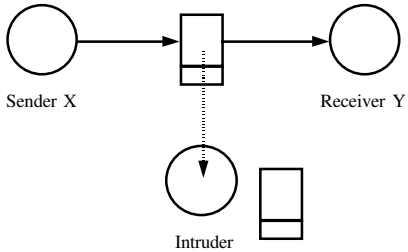


Discovering the Alteration

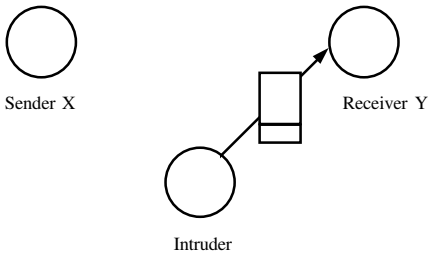


Z_S does not equal Z_{M_A}, so the signature is invalid

Replay Diagram



Replay Occurs



How To Handle This Replay?

- Sequence numbers in messages
- Challenge/response to sender
- Timestamp messages and discard "old" ones
- Don't worry about it

Major Challenge in Public Key Cryptography

How do I find out someone's public key?

If not done securely, the system is totally compromised

Must also be efficient

And how do I securely store and manage public keys?

Authentication Servers

Like key servers, trusted third parties

An authentication server can produce a ticket that guarantees the identity of a user

Generally tickets expire

Kerberos is the most popular authentication server

More on Kerberos

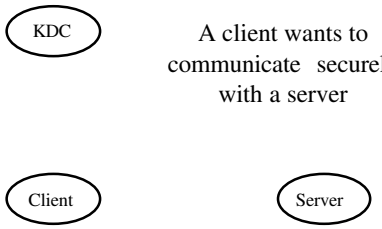
Uses symmetric cryptography

Servers are trusted by all parties

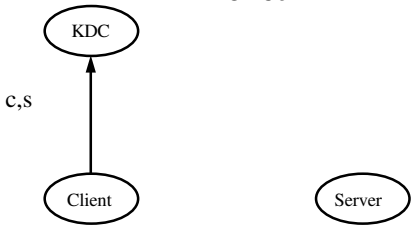
Issues tickets that provide secure communications between clients and servers

Tickets have a lifetime, then expire

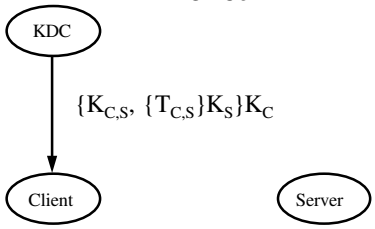
Kerberos in Action



The Client Asks Kerberos for a Ticket



Kerberos Gives the Client a Ticket



What's Going On Here?

What is in this message?

$T_{C,S}$ is the ticket that allows the client to communicate with the server

It's encrypted with K_S (so only the server can read it)

Message contains a new key $K_{C,S}$

Entire message encrypted in C's key

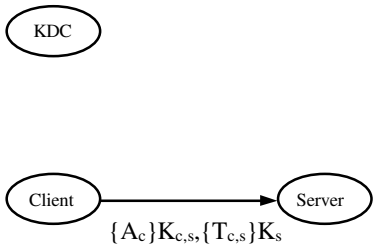
Why the Extra Key?

For authentication purposes

It's also contained within the ticket

Server can authenticate himself to client using that key

Client Sends Ticket to Server



What Does the Client Send?

Sends encrypted ticket from Kerberos server

Which only server can read

Also sends authenticator A_C in session key $K_{C,S}$

Server gets $K_{C,S}$ from ticket, sends back altered version encrypted with $K_{C,S}$

Firewalls

A program to allow selective access to the network

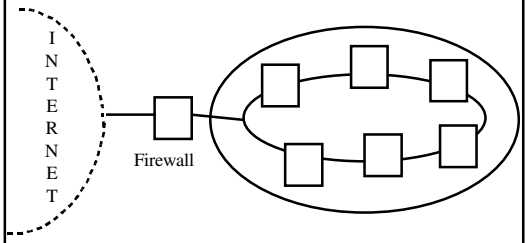
In both directions

Typically, firewalls protect entire networks

They must examine everything that tries to pass into the protected domain

Only authorized transmissions permitted

Firewall Example



What Do Firewalls Do Well?

Prevent intruders from accessing machines on your network

Prevent your users from inadvertently compromising security

What Do Firewalls Do Badly?

Prevent many forms of legitimate access

May get in the way of other forms of security

Often, there's no further security behind the firewall

So if it fails . . .