# Computer Security

## Introduction

Marius Minea

September 30, 2015

# What is this course about ?

Security of *systems*

- operating system + applications
- network security

# What is this course about ?

Security of *systems*

- operating system + applications
- network security

*Secure Programming*

- vulnerabilities and their prevention
- security of web applications

# What is this course about ?

Security of *systems*

- operating system + applications
- network security

*Secure Programming*

- vulnerabilities and their prevention
- security of web applications

*Cryptography*

- foundational for all of security

# What is this course about ?

Security of *systems*
- operating system + applications
- network security

*Secure Programming*
- vulnerabilities and their prevention
- security of web applications

*Cryptography*
- foundational for all of security

Security *protocols* and their modeling
- authentication, key generation/exchange, etc.
- principles and tools for modeling and analysis

# What is security?

"Security is [...] preventing adverse consequences from the intentional and unwarranted actions of others" [Bruce Schneier, *Beyond Fear*]

"Computer Security deals with the *prevention* and *detection* of *unauthorized* actions by users of a computer system" [D. Gollmann]

# What is security?

"Security is [...] preventing adverse consequences from the intentional and unwarranted actions of others" [Bruce Schneier, *Beyond Fear*]

"Computer Security deals with the *prevention* and *detection* of *unauthorized* actions by users of a computer system" [D. Gollmann]

A security system *prevents* attacks
possibly: detection, recovery, repair

# What is security?

“Security is [...] preventing adverse consequences from the intentional and unwarranted actions of others” [Bruce Schneier, *Beyond Fear*]

“Computer Security deals with the *prevention* and *detection* of *unauthorized* actions by users of a computer system” [D. Gollmann]

A security system *prevents* attacks
possibly: detection, recovery, repair

Security deals with *intentional* actions
incidental actions: *safety* ($\neq$ security !)

# What is security?

"Security is [...] preventing adverse consequences from the intentional and unwarranted actions of others" [Bruce Schneier, *Beyond Fear*]

"Computer Security deals with the *prevention* and *detection* of *unauthorized* actions by users of a computer system" [D. Gollmann]

A security system *prevents* attacks
  possibly: detection, recovery, repair

Security deals with *intentional* actions
  incidental actions: *safety* ($\neq$ security !)

*unauthorized* actions (from victim point of view); need not be illegal

# What is security?

"Security is [...] preventing adverse consequences from the intentional and unwarranted actions of others" [Bruce Schneier, *Beyond Fear*]

"Computer Security deals with the *prevention* and *detection* of *unauthorized* actions by users of a computer system" [D. Gollmann]

A security system *prevents* attacks
  possibly: detection, recovery, repair

Security deals with *intentional* actions
  incidental actions: *safety* ($\neq$ security !)

*unauthorized* actions (from victim point of view); need not be illegal

Implies the existence of an *attacker*, targeting *assets*
  thinking of/modeling attacker capabilities is essential
  incl. multiple, colluding attackers

# How to achieve security?

By knowing

tehnical details (operating systems, networks, programming, crypto)

# How to achieve security?

By knowing
tehnical details (operating systems, networks, programming, crypto)

By thinking
*security mindset* [v. Schneier]
like an attacker (technical *and* social aspects)
social engineering: e.g., impersonate maintenance to get access

# How to achieve security?

By knowing
  tehnical details (operating systems, networks, programming, crypto)

By thinking
  *security mindset* [v. Schneier]
  like an attacker (technical *and* social aspects)
    social engineering: e.g., impersonate maintenance to get access

By understanding:
  fundamental notions: what needs protected? how? from what attacks?
  principles (design/construction): general, not necessarily technical

# How to evaluate security?

[ B. Schneier, *Beyond Fear* ]

1. What *assets* are you trying to protect?
2. What are the *risks* to those assets?
3. How well does the solution *mitigate* those risks?
4. What *other risks* does the solution cause?
5. What *costs and compromises* does the solution impose?

# Security Objectives

**C***onfidentiality*

– protecting / hiding information or resources

– typically done through cryptography

  – or other undisclosed mechanisms

– not just *contents*, even *existence* may be confidential
  (cf. steganography)

– includes hiding the resources

**I***ntegrity*

**A***vailability*

# Security Objectives

*Confidentiality*

*Integrity*
= trust in data or resources
– expressed by preventing unauthorized modifications
We distinguish:
- data integrity (of *content*)
- data *origin* authentication

Integrity mechanisms
– *prevention* mechanisms
of unauthorized data manipulation (e.g. from outside)
of data manipulation in unauthorized ways (e.g. from inside)
– *detection* mechanisms
[M. Bishop: Computer Security: Art and Science, Pearson, 2003]

*Availability*

# Security Objectives

*Confidentiality*

*Integrity*

*Availability*
= the ability of using information or a resource in the desired way

A system which is not available can be worse than one nonexistent.

Availability is usually analyzed in the context of some (statistical) assumptions about the environment

if the assumptions are not satisfied, the system may be compromised

*denial of service attacks* – may be difficult to detect if the traffic (partially) matches the allowed statistic pattern

# Security objectives – other classifications

**P**

rivacy, **A**vailability-Authentication, **I**ntegrity, **N**on-repudiation

Parkerian Hexad (Donn Parker, 2002)

confidentiality
*posession/control* (important even without violating confidentiality)
integrity
*autenticity* (of origin or author)
availability
*utility* (ex. data converted to useless format $\neq$ disponibilitate)

# Other security objectives

[Handbook of Applied Cryptography]

signature
authorization
access control
timestamping
witnessing (by someone other than originator)
confirmation
anonymity
revocation
traceability / accountability

# Security Threats

Confidentiality, integrity, availability are *services* offered

We discuss (potential) *threats* and (real) *attacks* to those services

Threat classification [R. Shirey, cf. M. Bishop]
– disclosure
– deception (forcing acceptance of false data)
– disruption = interrupting / stopping normal service
– usurpation = unauthorized control of part of a system

# Threat mechanisms

Microsoft STRIDE threat model

*Spoofing* identity - impersonating

*Tampering* with data - falsifying / attack on integrity

*Repudiation* - negating the effect of an action

*Information disclosure* - attack to confidentiality

*Denial of service* - attack to availability

*Elevation of privilege* - unauthorized additional rights

# Threat Mechanisms

intercepie (snooping)
  in particular: (passive) wiretapping
modifying / altering data ⇒ deception
  also interruption / usurpation (gaining control)
  active wiretapping, man-in-the-middle attack
  (actively changing content)
impersonation (masquerading, spoofing)
repudiation of origin (e.g. in commercial transactions)
*denial of receipt* – a form of deception
delay – could be service interruption, also usurpation
*denial of service*

# Secure design principles

## Saltzer & Schroeder: The Protection of Information in Computer Systems, 1975

a) *Economy of mechanism*: keep the design as simple and small as possible
$\Rightarrow$ security by design, not as an afterthought

# Secure design principles

## Saltzer & Schroeder: The Protection of Information in Computer Systems, 1975

a) *Economy of mechanism*: keep the design as simple and small as possible
$\Rightarrow$ security by design, not as an afterthought

b) *Fail-safe defaults*: base access decisions based on permission rather than exclusion (default deny)

# Secure design principles

## Saltzer & Schroeder: The Protection of Information in Computer Systems, 1975

a) *Economy of mechanism*: keep the design as simple and small as possible
⇒ security by design, not as an afterthought

b) *Fail-safe defaults*: base access decisions based on permission rather than exclusion (default deny)

c) *Complete mediation*: check every access, every time
(including in exceptional cases, maintenance.)
NOT based on previously taken decisions

# Secure design principles

## Saltzer & Schroeder: The Protection of Information in Computer Systems, 1975

a) *Economy of mechanism*: keep the design as simple and small as possible
$\Rightarrow$ security by design, not as an afterthought

b) *Fail-safe defaults*: base access decisions based on permission rather than exclusion (default deny)

c) *Complete mediation*: check every access, every time
(including in exceptional cases, maintenance.)
NOT based on previously taken decisions

d) *Open design*: (NOT: security through obscurity)
$\Rightarrow$ mechanisms may be publicly checked to gain trust

# Saltzer and Schroeder (cont.)

e) *Separation of privilege*: separation increases robustness

# Saltzer and Schroeder (cont.)

e) *Separation of privilege*: separation increases robustness

f) *Least privilege*: every program and user should operate with the minimal set of privileges needed for the given task

# Saltzer and Schroeder (cont.)

e) *Separation of privilege*: separation increases robustness

f) *Least privilege*: every program and user should operate with the minimal set of privileges needed for the given task

g) *Least common mechanism*: minimize common resources, interference among users, the mechanisms on which everything is based

# Saltzer and Schroeder (cont.)

e) *Separation of privilege*: separation increases robustness

f) *Least privilege*: every program and user should operate with the minimal set of privileges needed for the given task

g) *Least common mechanism*: minimize common resources, interference among users, the mechanisms on which everything is based

h) *Psychological acceptability*:
not unduly interfere with common activity
if mechanisms are not simple, they will be misused or bypassed

# Saltzer and Schroeder (cont.)

e) *Separation of privilege*: separation increases robustness

f) *Least privilege*: every program and user should operate with the minimal set of privileges needed for the given task

g) *Least common mechanism*: minimize common resources, interference among users, the mechanisms on which everything is based

h) *Psychological acceptability*:
not unduly interfere with common activity
if mechanisms are not simple, they will be misused or bypassed

2 additional ones:
*Work factor*: compare needed effort with attacker resources
*Compromise recording*: in case of failure, an alarm still useful

# Security principles (cont.)

weakest link determines security of entire system

adequate protection principle
   not maximal security, but utility at acceptable risk/cost

principle of efficiency (cf. acceptability)
   appropriate, easy to use correctly

defense in depth: layered protection

[Ninghui Li, CS 426: Computer Security, course, Purdue University]

# Attack Actions

– “probe”: acces a target to determine characteristics
– “scan”: sytematically access (probe) several targets
– “flood”: repeated access to a target to overload it
– authentication: present an identity for verification and ulterior access
– bypass: circumvent a control/authorization process using an alternate method to access a target
– spoof/masquerade: assume some other identity
– read
– copy
– steal (take into posession and eliminate the original)
– modify
– delete

# Result of an attack

unauthorized (increased) access to a system or network

information disclosure (attack to confidentiality)

information corruption (atac la integritate)

denial of service (attack to availability)

theft of resources (unauthorized use): a type of usurping resource

# Security: general problems [Schneier]

error modes: passive vs. active (does not vs. does what it shoudln't)

danger of errors in rare cases

*security imbalances* – effect of large-scale technologies

fragile (brittle) systems vs. resilient to errors

protection methods: *adaptive* to unforeseen situations

monocultures (homogeneous systems) – vulnerable to same attack
 e.g. majority of systems is running Windows...

securitatea e o problem uman / social

# Security and Trust

In security, we make *assertions* (statements) of various entities

These statements are not *absolute*, they are based on *assumptions*.

$\Rightarrow$ Security is a matter of trust: in whom/what can we trust?

Ken Thompson: Reflections on Trusting Trust (Turing Award Lecture '83)
inserted a *trojan* into the `login` program and C compiler
to accept a special password (known by originator)
by using *self-reproducing* code

"You can't trust code that you did not create yourself"

"No amount of source-level verification or scrutiny will prevent you from using untrusted code"

# Example: file protection in Unix (review)

every file is *owned* by a user and group

individual permission bits: **r**ead, **w**rite, e**x**ecute/search

3 groups of bits for: **u**ser, **g**roup, **o**thers

Meaning for *directories* is more complex than for files:

**r** is needed for `read()`, `readdir()`, `opendir()` $\Rightarrow$ for `ls`

**x** ("search") is needed for `chdir()` and `stat()` (any file)

# Unix file permission examples

What permissions are needed to read a file ?

# Unix file permission examples

What permissions are needed to read a file ?

- **x** on the entire path and **r** for the file

# Unix file permission examples

What permissions are needed to read a file ?

- **x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

# Unix file permission examples

What permissions are needed to read a file ?

**x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

needs info from *inode*, thus `x` on the parent directory (also, `x` on the path); independent of permissions on *file*.

if *file* is a directory, `ls -l` lists contents (needs `r`)

`ls -ld` only gives directory info, so answer is as above

# Unix file permission examples

What permissions are needed to read a file ?

**x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

needs info from *inode*, thus `x` on the parent directory
(also, `x` on the path); independent of permissions on *file*.

if *file* is a directory, `ls -l` lists contents (needs `r`)
`ls -ld` only gives directory info, so answer is as above

What permissions are needed to delete a file ?

# Unix file permission examples

What permissions are needed to read a file ?

**x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

needs info from *inode*, thus `x` on the parent directory (also, `x` on the path); independent of permissions on *file*.

if *file* is a directory, `ls -l` lists contents (needs `r`) `ls -ld` only gives directory info, so answer is as above

What permissions are needed to delete a file ?

**w** in parent directory, as well as `x`

Need not have `w` for the file!

# Unix file permission examples

What permissions are needed to read a file ?

**x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

needs info from *inode*, thus `x` on the parent directory
(also, `x` on the path); independent of permissions on *file*.

if *file* is a directory, `ls -l` lists contents (needs `r`)
`ls -ld` only gives directory info, so answer is as above

What permissions are needed to delete a file ?

**w** in parent directory, as well as `x`

Need not have `w` for the file!

What can you do with **x** on directory but not `r` ?

# Unix file permission examples

What permissions are needed to read a file ?

**x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

needs info from *inode*, thus `x` on the parent directory (also, `x` on the path); independent of permissions on *file*.

if *file* is a directory, `ls -l` lists contents (needs `r`)

`ls -ld` only gives directory info, so answer is as above

What permissions are needed to delete a file ?

**w** in parent directory, as well as `x`

Need not have `w` for the file!

What can you do with **x** on directory but not `r` ?

You can access a file with known name, but can't search for a file

(e.g. search for file on a web server)

# Unix file permission examples

What permissions are needed to read a file ?

**x** on the entire path and **r** for the file

What permissions are needed for `ls -l` *file*?

needs info from *inode*, thus `x` on the parent directory
(also, `x` on the path); independent of permissions on *file*.

if *file* is a directory, `ls -l` lists contents (needs `r`)
`ls -ld` only gives directory info, so answer is as above

What permissions are needed to delete a file ?

**w** in parent directory, as well as `x`

Need not have `w` for the file!

What can you do with **x** on directory but not `r` ?
You can access a file with known name, but can't search for a file
(e.g. search for file on a web server)

Special bits:
– sticky bit: for directory: file can only be deleted by owner
– set user ID: execute with *effective* ID of file owner
– set group ID: execute with *effective* ID of file group