NANYANG TECHNOLOGICAL UNIVERSITY

Lecture 1: Software Security

presented by

Li Yi
Assistant Professor
SCSE

N4-02b-64
yi_li@ntu.edu.sg

COPYRIGHT STATEMENT

- All course materials, including but not limited to, lecture slides, handout and recordings, are for your own educational purposes only. **All the contents of the materials are protected by copyright, trademark or other forms of proprietary rights.**
- All rights, title and interest in the materials are owned by, licensed to or controlled by the University, unless otherwise expressly stated. **The materials shall not be uploaded, reproduced, distributed, republished or transmitted in any form or by any means, in whole or in part, without written approval from the University.**
- You are also not allowed to take any photograph, film, audio record or other means of capturing images or voice of any contents during lecture(s) and/or tutorial(s) and reproduce, distribute and/or transmit any form or by any means, in whole or in part, without the written permission from the University.
- Appropriate action(s) will be taken against you including but not limited to disciplinary proceeding and/or legal action if you are found to have committed any of the above or infringed the University's copyright.

About Me

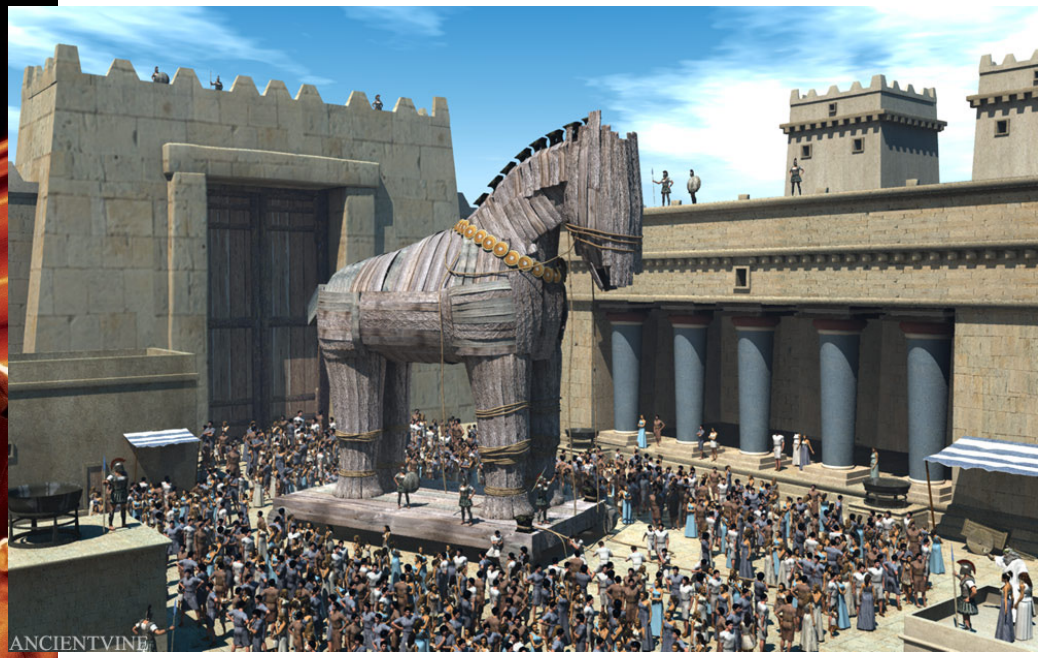
- A Brief History
  - B.Comp. (hons) in Computer Science, NUS, Singapore
  - M.Sc. & Ph.D. in Computer Science, University of Toronto, Canada

- Research Interests
  - Program analysis, automated reasoning
  - Software verification
  - Software reuse
  - Software security

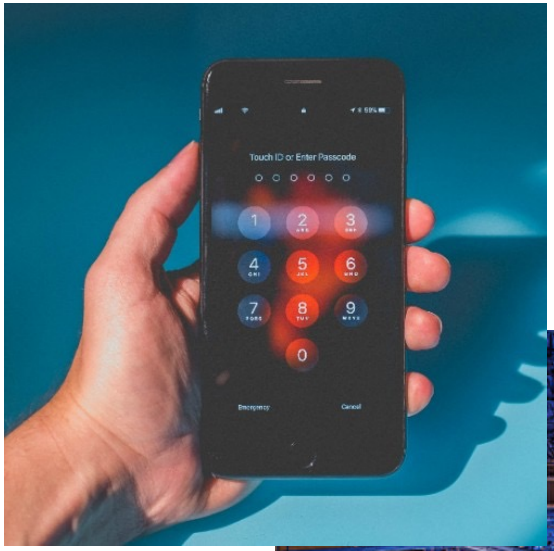
- Contacts
  - Office: N4-02c-64
  - Email: yi_li@ntu.edu.sg

What is Security?

What is Security?



Computers are Used Everywhere



A system complex enough to be interesting cannot be fault free

Jean-Claude Laprie, formerly at LAAS, Toulouse

Computer Security

"What you must learn is that these rules are no different than the rules of a computer system ... some of them can be bent. Others ... can be broken. Understand?"

-- The Matrix (1999)

Computer Security

- Computer security in a nutshell:
  - Computer systems have rules: some are enforced but others are implicit because designers made unwritten assumptions.
  - Security vulnerabilities occur when those assumptions turn out to be false – something happens that the programmer did not expect.
- Examples:
  - An attacker can inject and execute arbitrary code on a system (buffer overflow)
  - An attacker can forge an e-mail to make it look like it came from someone else (targeted spam)
  - An attacker can extract keys from a hardware chip (differential power analysis)

Computer Security is about understanding a system really well so you can anticipate the unexpected.

Computer Security

Cyber attacks cost **real** money, do **real** damage

- Cost:
  - $100 million
  - Leaked films, cancelled openings, …

- Cost:
  - $148 million
  - CEO and CTO's jobs

- Cost:
  - 1.5 million patients had their personal particulars stolen
  - IHiS, SingHealth fined $1m

Computer Security

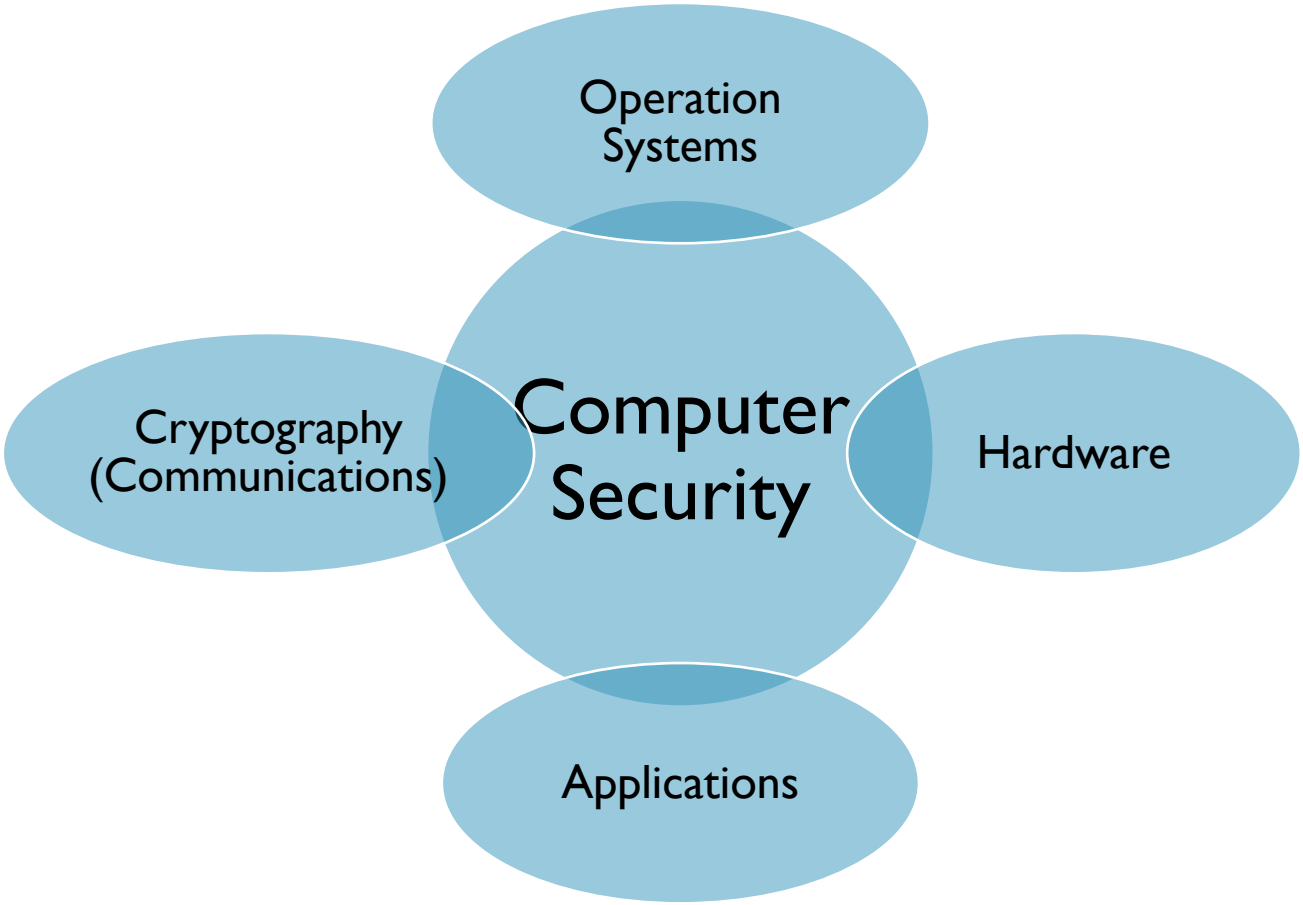
Cyberspace is a military and political tool

Computer Security

Why learn about computer security?

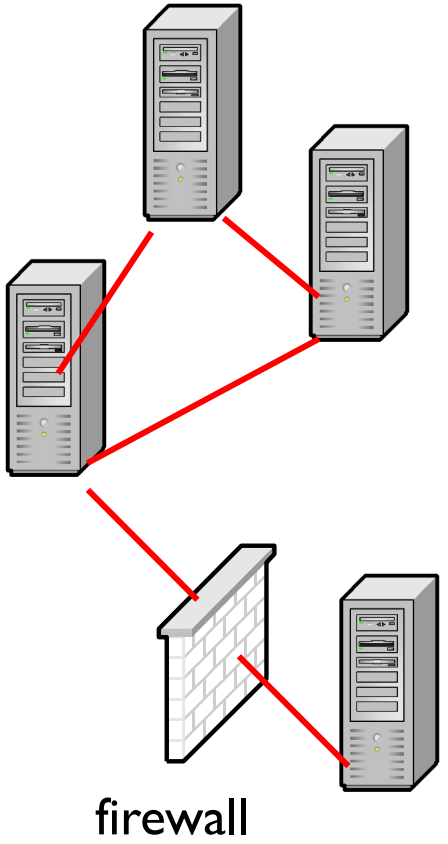
- It's important
  - Microsoft releasing "Black Tuesday" patches
  - Android releasing monthly patches
- There's a lot of money involved
  - Target, Home Depot, Sony = $100s Ms
  - Zero day vulnerabilities regularly sell for $10K's to $Ms
- You use computers
  - You have an Android, iPhone or some other smartphone
  - You have a PC with an internet connection
- You're curious …

The Big Picture



Aspects of Security

- Communications (network) security: addresses security of the communications links
- Computer security: addresses security of the end systems; today, this is the difficult part; software security fits in here
- Application security: relies on both to provide services securely to end users
- Security management: how to deploy security technologies



Distributed systems: computers connected by networks

The Big Picture

- In the 1980s, security research and practice was dominated by operating system security
- In the 1990s, security research and practice was dominated by communications security (SSL/TLS, IPsec, firewalls, …)
- From the mid 1990s onwards, most vulnerabilities reported happened to be software security problems
- Software security is hardly covered in traditional academic programs; this course intends to fill this gap, create awareness of the problems software developers are likely to encounter, and give an introduction to tools and countermeasures

Software Security vs. Application Security

- Software Security: defends against exploits by building software to be secure in the first place, by getting the design right and avoid common mistakes. Issues include:
  - Software risk management, programming languages and platforms, auditing software, security by design, security flaws (buffer overflows, race conditions, access control and password problems, cryptographic errors, etc.) and security testing

- Application Security: defends against exploits after development and deployed. Issues include:
  - Authentication, integrity checks, sandboxing code, protection against mobile malicious code, runtime monitoring and enforcement of security policies

Course Objectives

- Understand software security and its implications
- Examine major causes for software vulnerabilities
- Examine current practices for addressing these problems
  - Help you avoid "unforgivable" vulnerabilities in the code you write & know how to mitigate
  - http://cve.mitre.org/docs/docs-2007/unforgivablevulns_bh.pdf
- We intend to give code examples but will not try to teach you how to write worms and viruses

Prerequisites

- CX2002 & CX2005
- Some familiarity with C (but you will not be asked to code in C); if you have no experience in programming at all, this course will be difficult
- Some familiarity with the basics of operating systems (mainly Linux): user mode vs kernel mode, stack, heap, relative addressing, …
- Ideally, also some understanding of the foundations of programming languages, e.g., type safety
- The willingness to search the web for further information and the willingness to engage with us

Rules of Engagement

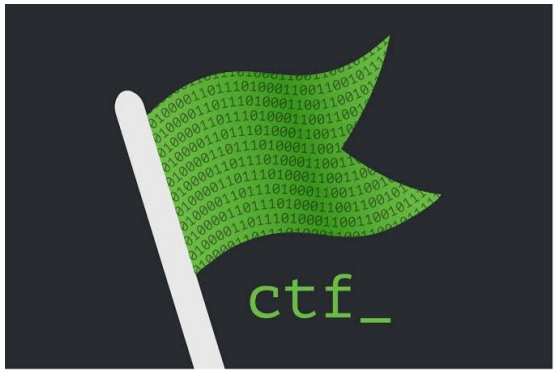
- If something is unclear during the lectures, don't hesitate to ask
- If – despite our best efforts – you find an error in the lecture material, please correct us
- If you have additional knowledge, please share your expertise with the rest of the class

Don't try it at home

- Distributing code that performs actions on other peoples machines is likely to get you in conflict with the law
- Experimenting with dangerous code at home may be an intellectual challenge but is fraught with danger
- Anti-virus researchers learned at their own expense the importance of physically separating experimental from operational systems

Do try it at home

- Many fun (!!) exercises online
- Often called 'CTFs' or 'hackmes'
  - (older) sometimes 'wargames'
  - More on this later…
- Some good ones:
  - https://www.hackthebox.eu/
  - https://overthewire.org/
  - http://pwnable.kr/
  - https://microcorruption.com/



Structure of the Course

- 60% of marks through final examination (absence of exam will fail the course!!!)
- 20% of marks through quiz (week 8 tutorial)
  - ~20 multiple choice questions, 40 mins
  - No Makeup Quiz!!!
- 20% of marks through hands-on exercises (our 4067 CTF competition!)
  - Team size: 1~3
  - Week 10: *20 March -- 27 March*
  - Report due Week 13
- Refer to course calendar (tentative) for schedule details

4067 CTF Challenge!

- Put your knowledge into practice
- Prizes for winners (shopping vouchers)!
- Scoring rules to be announced
- Form a team early: team registration opens from Week 9

Prize	Award Value (per team)
1st Prize	S$100
2nd Prize	S$70
3rd Prize	S$50
4th – 9th	S$20

Course Contents

1. General introduction (this lecture)
2. Assessing/measuring security
3. Buffer overflows; DEP, ASLR, JOP, ROP
4. Integer overflows, targeted overwrites
5. Other vulnerabilities, memory residues
6. Race conditions
7. Meta characters, Unicode bugs (E-learning)
8. Scripting languages: SQL inserts, cross-site scripting attacks
9. Generic countermeasures: type safety, regular expressions, taint analysis
10. Secure software development; security testing

Books

- Michael Howard & David LeBlanc: Writing Secure Code, Microsoft Press, 2nd edition, 2002
  - E-book available on NTULibrary
- Gary McGraw: Exploiting Software: How to Break Code, Addison-Wesley, February 2004
- John Viega & Gary McGraw: Building Secure Software, Addison-Wesley, 2001
- J.C. Foster et al.: Buffer Overflow Attacks, Syngress, 2005
- Mark G. Graff & Kenneth R. van Wyk: Secure Coding, O'Reilly, 2003

References

For information on recent vulnerabilities see, e.g.:

- CERT advisories http://www.cert.org/advisories/, advisories from national security response teams.
- seclists.org (BugTraq, Full Disclosure)
- SANS institute: http://www.sans.org/
- Web sites of software vendors,
  - E.g. Microsoft security advisories: http://www.microsoft.com/technet/security/advisory/default.mspx
  - Mozilla: http://www.mozilla.org/security/known-vulnerabilities/
- Web sites of security companies: Kaspersky, McAfee, Sophos, Symantec, …

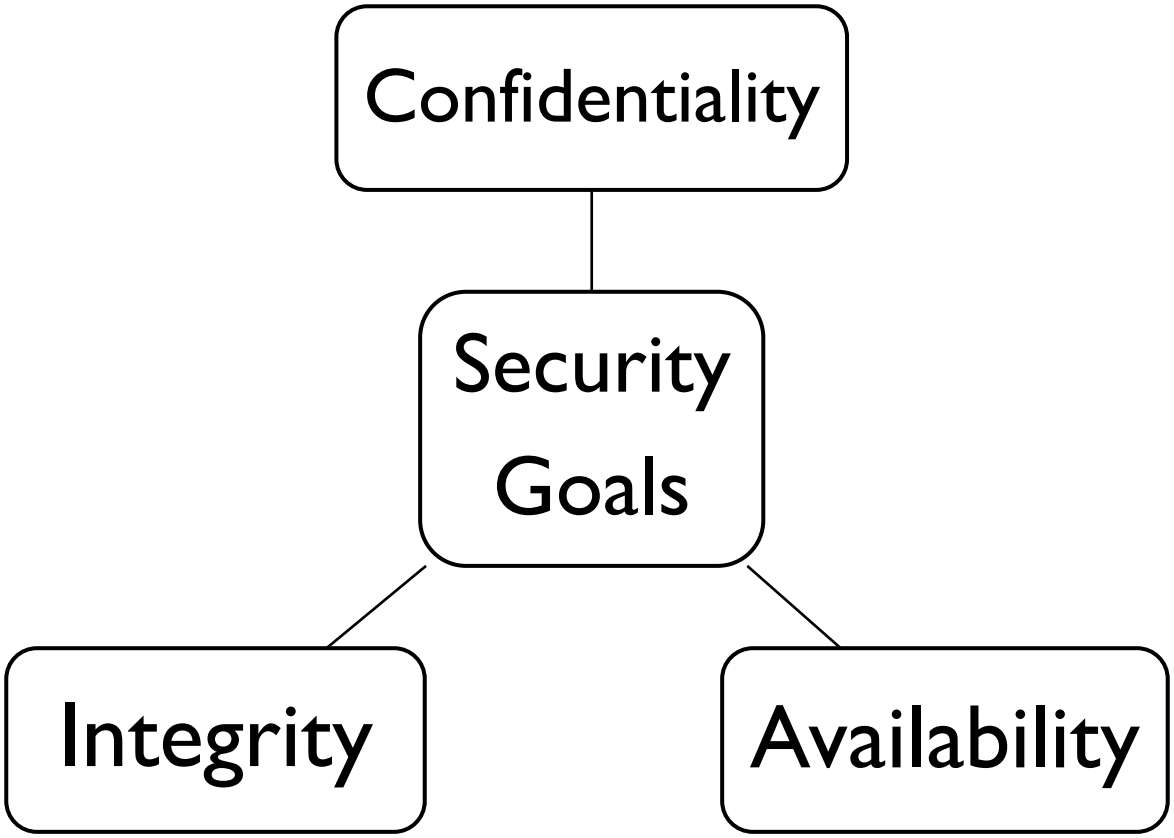
Introduction to Software Security

Agenda

- Security Fundamentals
  - Security components & goals
  - Aspects of computer security
  - Malware
  - Vulnerabilities
  - Security strategies
- Secure software
  - Why secure software
  - Design for security

Security Fundamentals

Components of Security



Confidentiality

"The protection of information or resources from exposure."

- Also commonly referred to as *secrecy*.
- There are two aspects of information or resources that are often important to conceal:
  1. The *content* of a piece of information or resource. E.g.: most people want to keep their NTU account passwords confidential. In many cases, content may also cover things such as configuration, cost or other attributes.
  2. The *existence* of information or resource. In this case, the mere knowledge that something exists is damaging. E.g.: a company may want to keep the existence of a new product secret until it is ready for commercial introduction to the market.

Integrity

"The trustworthiness of information or resources."

- Integrity tells you how ready you should be to believe something is correct. Like confidentiality, there are two aspects of integrity that are important:
  1. The correctness of the *contents* of a piece of information or resource. E.g.: by altering the contents of a piece of e-mail, you are violating the integrity of its contents.
  2. The correctness of the *origin* of a piece of information. E.g.: by altering the sender's e-mail address, an attacker makes an e-mail look like it's coming from someone else. Verifying the source of a request or some data is called *authentication*.

Availability

"The ability to access or use information or resources as desired."

- A resource is available if it is accepting and responding to requests. Information is available if the service which stores that information is up and running. Availability is generally more difficult to deal with for two reasons:
  1. In automata theory, availability is not a *finite* property. Unlike confidentiality or integrity, you can't always say that at time *t* some object has become unavailable.
  2. Many systems deal with availability in a probabilistic fashion. Components in any system have some inherent level of unreliability and the unreliability of the system is the composition of unreliability of these components. However, availability cannot be treated probabilistically for security since an active adversary will cause unlikely events to occur more often (i.e. network flooding).

CIA = Security

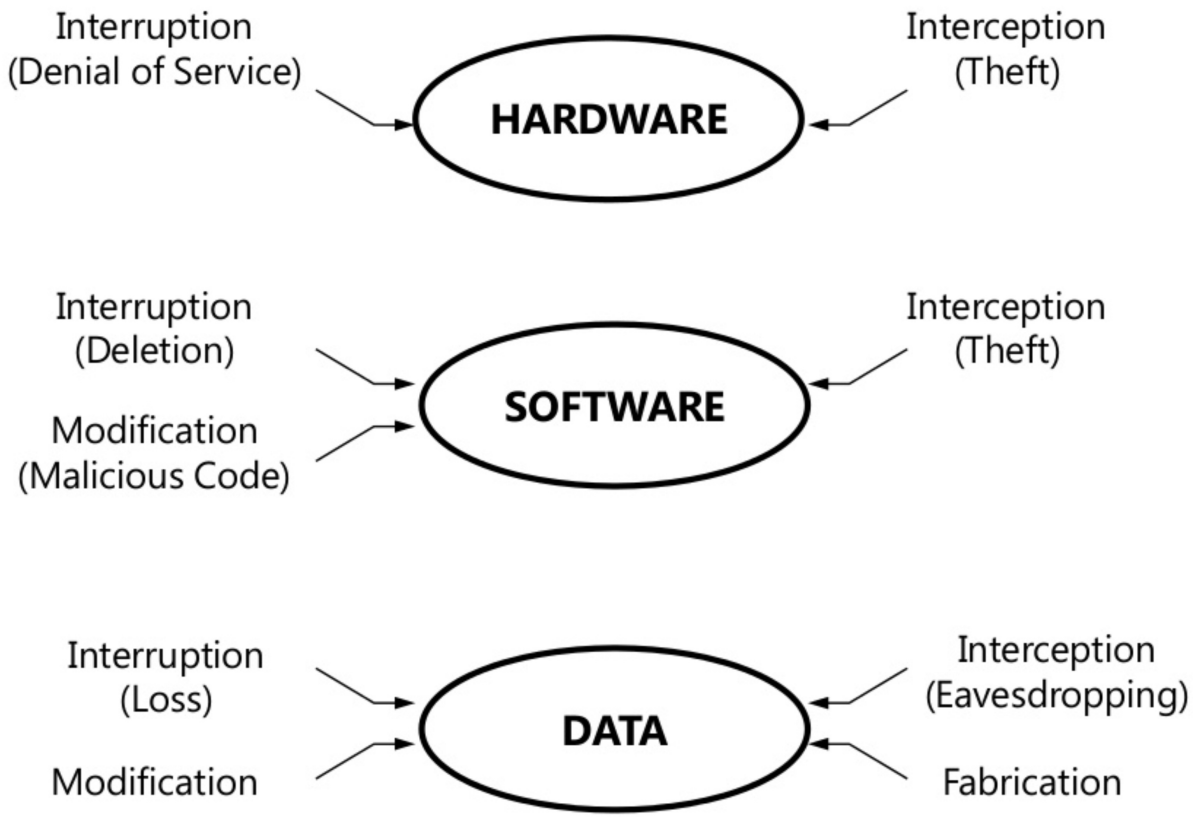
- Any system that can be called secure provides all three (**C**onfidentiality, **I**ntegrity & **A**vailability) of these attributes to some degree. Remember that no system is absolutely secure, and so no system can provide all or any of the three absolutely.
- Measures (more in Lecture 2):
  - Confidentiality & Integrity are often provided by cryptographic algorithms. Their strength is often measured in terms of complexity (how long will it take to break the algorithm). E.g.: 256-bit keys are considered more secure than 128-bit keys.
  - Availability is very hard to measure quantitatively. Traditional measures (probabilistic) measure availability in terms of percentage of time a system is accessible. A system with 99.999% (five 9's) availability is only down 0.001% of the time. Unfortunately, this doesn't apply well to measuring security.

Kinds of Security Breaches

- The major assets of computing systems are *hardware*, *software*, and *data*.

- There are four kinds of threats to the security of a computing system:
  - Interruption: an asset of the system becomes lost, unavailable, or unusable.
  - Interception: some unauthorized party has gained access to an asset.
  - Modification: an unauthorized party not only accesses but tampers with an asset.
  - Fabrication: an unauthorized party might make counterfeit objects on a computing system.

Types of Vulnerabilities



Threats to Software

- Software Deletion
  - By accident or intentionally
- Software Modifications
  - Trojan horse: a program that overtly does one thing while covertly doing another
  - Virus: a specific type of malware, that can be used to spread infection from one computer to another
  - Trapdoor: a program that has a secret entry point
  - Information leaks: make information accessible to unintended people or programs
  - More on malware later…
- Software Theft
  - Unauthorized copying of software

More subtle than threats to hardware

Malware

Malware: general term for malign software

- Virus: self-replicating code that spreads by embedding itself in executable files or system areas in memory
- Worm: self-contained self-replicating program; need not be part of another program to propagate itself
- Trojan horse: malign program disguised as legitimate software; usually not intended to replicate itself (spyware, adware, …)

History of Malware

- 1982: Xerox worm program
  - John F. Shoch & Jon A. Hupp: The "worm" programs – early experience with a distributed computation, Communications of the ACM 25(3), pp. 172-180, March 1982
- 1982: first Macintosh viruses
- 1983: Fred Cohen
  - Demonstrated a virus on a VAX 11/750 system running Unix, at a security seminar on November 10, 1983
  - Fred Cohen: Computer Viruses, in Computer Security: A Global Challenge, Elsevier Press, pp. 143-158, 1984
- 1986: Brain – earliest known virus for MS-DOS
  - First reported at the University of Delaware in January 1986; boot-sector virus infecting DOS formatted 360K floppy disks

Morris Worm (2-Nov-1988)

- One of the first computer worms distributed via the Internet
- Penetrated 5--10% of the machines on the Internet (≈ 60,000), exploited known vulnerabilities
  - Bad configurations: debug mode in `sendmail`
  - Buffer overflow in `fingered` daemon
  - Brute force password guessing for remote login
  - `rsh` vulnerability: unauthenticated login from trusted hosts identified by their network address (which could be forged)
- Worm was intended to propagate slowly and harmlessly measure the size of the Internet
- $10-100M worth of damage

Image credit: https://en.wikipedia.org/wiki/Morris_worm

Morris Worm Cont'd

- Serious mistake in reaction to the worm: turn off `sendmail` → other nodes not warned in time
- Graduate student at Cornell, son of NSA chief scientist
- Convicted under Computer Fraud and Abuse Act
  - The first felony conviction under this act
  - Court sentence (May 4, 1990): three years probation, $10,050 fine, 400 hours community service

History of Malware – 2 (4-Oct-2005)

- Samy MySpace Worm
  - Most infamous cross-site scripting (XSS) worm
  - Users can post HTML on their pages
  - MySpace.com tries to filter out `<script>`, `<body>`, `onclick`, `<a ref=javascript://>`
  - However, they neglected to filter out javascript in CSS tags: `<div style="background:url('javascript:alert(1)')">`
  - And one can hide javascript with a line break as `java\nscript`

Watch the video:

https://www.youtube.com/watch?v=DtnuaHl378M

History of Malware – 2 Cont'd

- Samy's worm: infects anyone who visits an infected MySpace page and adds Samy as a friend.
  - Samy had millions of friends within 24 hours
  - In this case MySpace is harmed as this effectively slows down MySpace by consuming resources
  - Sentenced three years probation, no use of Internet, 90 days community service
- More Info:
  - https://samy.pl/myspace/tech.html

History of Malware – 3 (12-May-2017)

- WannaCry
  - Ransomware / cryptoware
  - Asks payment using Bitcoin
  - Targets at MS Windows – exploit vulnerability in the Server Message Block (SMB) protocol
  - Likely known to NSA prior to the attack
  - Microsoft issued security bulletin MS17-010 and patches on March 14, 2017

  - Affected more than 200,000 computers across 150 countries
  - Total damages ranging from hundreds of millions to billions of dollars

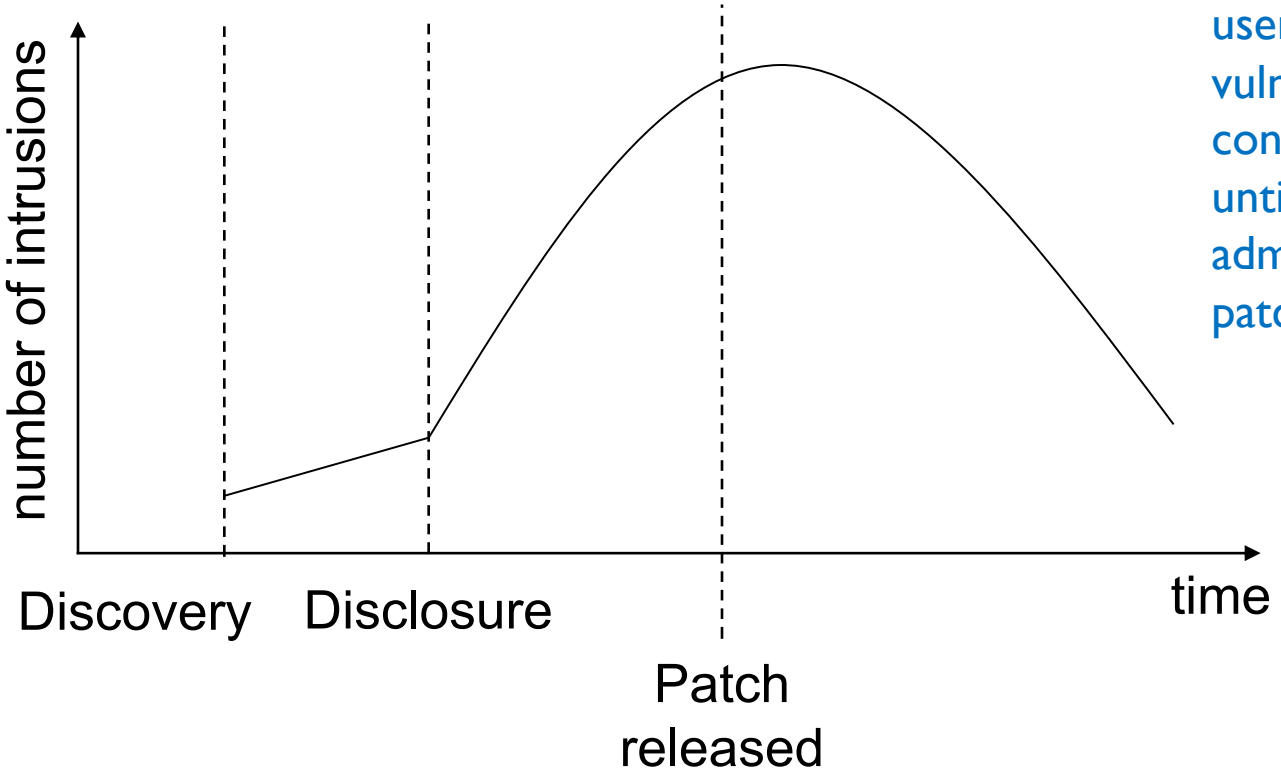
Attack Scripts

- Vulnerabilities in code are nothing new
- Systematic vulnerability search started around 1996
  - Aleph One: Smashing The Stack For Fun And Profit, Phrack, 1996
- Tool support available to search for certain types of vulnerabilities
- Attack scripts for exploiting software flaws (e.g., Metasploit)
  - So-called script kiddies can launch attacks without having to understand how they work

Vulnerability Disclosure

- When should vulnerabilities be disclosed?
  - When they are found, after a patch is available, or never?
- Moral arguments:
  - Disclosure forces software companies to improve
  - Will unsophisticated end users suffer?
- Economic argument: does it worth it?
  - The vulnerability identifier has the incentive to leak the discovery (in unregulated market)
  - *Karthik Kannan, Rahul Telang: Economic Analysis of the Market for Software Vulnerability Disclosure, Proceedings of the 37th Hawaii International Conference on System Sciences, 2004*
- Some guidelines to follow https://cheatsheetseries.owasp.org/cheatsheets/Vulnerability_Disclosure_Cheat_Sheet.html

Average number of intrusions for a vulnerability over time



Intrusions increase once users discover a vulnerability, and the rate continues to increase until the system administrator releases a patch or workaround

W. Arbaugh, B. Fithen, J. McHugh: Windows of Vulnerability: A Case Study Analysis, IEEE Computer, 12/2000

Security Strategies

- Prevention: take measures that prevent your assets from being damaged
- Detection: take measures so that you can detect when, how, and by whom an asset has been damaged
- Reaction: take measures so that you can recover your assets or to recover from a damage to your assets

Countermeasures for Vulnerabilities

- Prevention
  - Avoid vulnerabilities in new code
  - Eliminate vulnerabilities from existing code base
  - Harden execution environment so that attempts to exploit vulnerable code are stopped
- Detection & reaction
  - Virus/malware scanners
  - Canaries (run-time checks)
  - Intrusion detection systems (intrusion response systems)

Penetrate-and-Patch is Bad

- Developers can only patch problems which they know about. Attackers may find problems that they never report to developers.
- Patches are rushed out as a result of market pressures on vendors, and often introduce new problems of their own to a system.
- Patches often only fix the symptom of a problem, and do nothing to address the underlying cause.
- Patches often go unapplied, because system administrators tend to be overworked and often do not wish to make changes to a system that "works".
- Designing a system for security, carefully implementing the system, and testing the system extensively before release, presents a much better alternative.

Secure Software

First, why people choose not to build secure systems?

- Security is boring
- Security is usually not the primary skill or interest of the designers and developers creating the product
- Security means not doing something exciting and new
- Security is often seen as a functionality disabler, as something that gets in the way
- Security is difficult to measure

Arguments on why secure software!

1. **Secure products are quality products**
   - Not as simple, as perfect security is an oxymoron. The most secure system is the one that's turned off
   - Argue about software secure enough and good enough for the environment in which it will operate
   - Although context-driven, what's clear in this argument is that security is a subset of quality. A product that is not appropriately secure is inferior to competing products

Arguments on why secure software!

2. **The media (and your competition) leap on security issues**
   - No better news as bad news
   - Press loves to make headlines out of security problems; often they don't really know what they're talking about and mischaracterize or exaggerate
   - Serious or not, you don't want your products in the headlines due to a security issue
   - Competitors will exploit this and market their more "secure products"

Arguments on why secure software!

3. **People shy away from products that don't work as advertised**
   - People will begin to shy away and start looking for solutions from competitors
   - People who have grudge against your product will fan the fire
   - It's an unfortunate human trait, but people tend to keep track of information that complies with their biases and agendas

Arguments on why secure software!

4. **Don't be a victim**
   - A misguided belief in the market that people who can break into systems are also the people who can secure them
   - Hence, there are a lot of would-be consultants who believe that they need some trophies mounted on their wall for people to take them seriously
   - You don't want your product to be a head on someone's wall!

Arguments on why secure software!

5. Security vulnerabilities are expensive to fix

- Like all engineering changes, security fixes are expensive to make late in the development process
- The price of making one includes the following:
  - Cost of the fix coordination. Someone has to create a plan to get the fix completed
  - Cost of developers finding and fixing the vulnerable code
  - Cost of testers testing the fix
  - Cost of creating and testing international versions

Security vulnerabilities are expensive to fix

- The cost of digitally signing the fix if you support signed code
- The cost of posting the fix to your website and writing the supporting documentation
- The cost of handling bad public relations
- Bandwidth and download costs if you pay an ISP to host fixes for you
- The cost of lost productivity. Chances are good that everyone involved in this process should be working on new code instead. Working on the fix is time lost

Security vulnerabilities are expensive to fix

- The cost to your customers to apply the fix. They might need to run the fix on a nonproduction server to verify that it works as planned. Productivity lost!
- Finally, the potential cost of lost revenue, from likely clients deciding to either postpone or stop using your product

- If only you had had security in mind when you designed and built the product in the first place!!!

Costing a security bug

- It's hard to determine a dollar cost for a fix because there are many intangibles
- While it is difficult to determine the exact cost of issuing a security fix, the Microsoft Security Response Center believes a security bug that requires a security bulletin costs in the neighborhood of $100,000
- Take a look at this: https://www.justice.gov/criminal-ccips

Secure Software

- Software is secure if it can handle intentionally malformed input; the attacker picks (the probability distribution of) the inputs
- Secure software ≠ software with security features
  - Security is not a feature you can add to a system at any time.
  - Security is a behavioural property of a complete system in a particular environment.
  - It is always better to design for security from scratch than try to add security to an existing design.
- The environment problem
  - A system that is secure enough in one environment is completely insecure when placed in another.
  - E.g., network applications moved from proprietary networks to the Internet.

Security & Quality

- Software security can be treated as an aspect of software quality
- Better quality software has fewer bugs
- Comment from the early 1990s:
  - *Japanese software companies lost out to US competitors because they had put too much emphasis on quality*
- The market was more keen on features then
  - *"The customer is always right"* (but may regret it later)
  - Another problem: it is difficult to measure security (more on Lecture 2)

Security vs. Reliability

- Reliability deals with accidental failures: failures are assumed to occur according to some given probability distribution
- The probabilities for failures are given first, then the protection mechanisms are constructed
- To make software more reliable, it is tested against typical usage patterns:
  - *"It does not matter how many bugs there are, it matters how often they are triggered"*

Security vs. Reliability

- In security, the defender has to move first; the attacker picks inputs to exploit weak defences
- To make software more secure, it has to be tested against "untypical" usage patterns (but there are typical attack patterns)
  - On a PC, you are in control of the software components sending inputs to each other
  - On the Internet, hostile parties can provide input: Do not "trust" your inputs

Design for Security

- Security should be considered during all phases of the development cycle and should deeply influence a system's design
- Frequent advice: "security must be considered early on in the design process"
- Else, design decisions already made may make it difficult to achieve the security goals
  - Win 95/98/ME: not designed for today's networking environment
- Things to think about during design (security model)
  - How data flow between components
  - Any users, roles, and rights, either explicitly stated or implicitly included in the design
  - The trust relationships of each component
  - Any potentially applicable solution to a recognized problem

Design for Security

- It is in general not possible to achieve security by having a "secure" design
  - Flaws my be introduced in the implementation
- Analysing the design of a software system is a good way of looking for security bugs
  - You don't have to analyse source code to find bugs in software
  - 50% of security bugs found in the design phase [Howard & LeBlanc]

Security Relevant Code

- A suggestive strategy: identify the "security relevant code", make sure that you get it right
- Unfortunately, this is another misconception. In practice, much more code than one would expect turns out to be security relevant
  - Microsoft Clip Gallery 5.0 (buffer overrun)
  - JPEG vulnerabilities (JPEG-of-Death)
  - Japanese character sets in cross site scripting problems
  - Characters terminating user inputs, file names
  - Unix sort DoS vulnerability (US-CERT VU#417216)
  - zip archiving utility
  - …

Tutorial – Next Monday

- Tutorial 1: Side Channels and Covert Channels