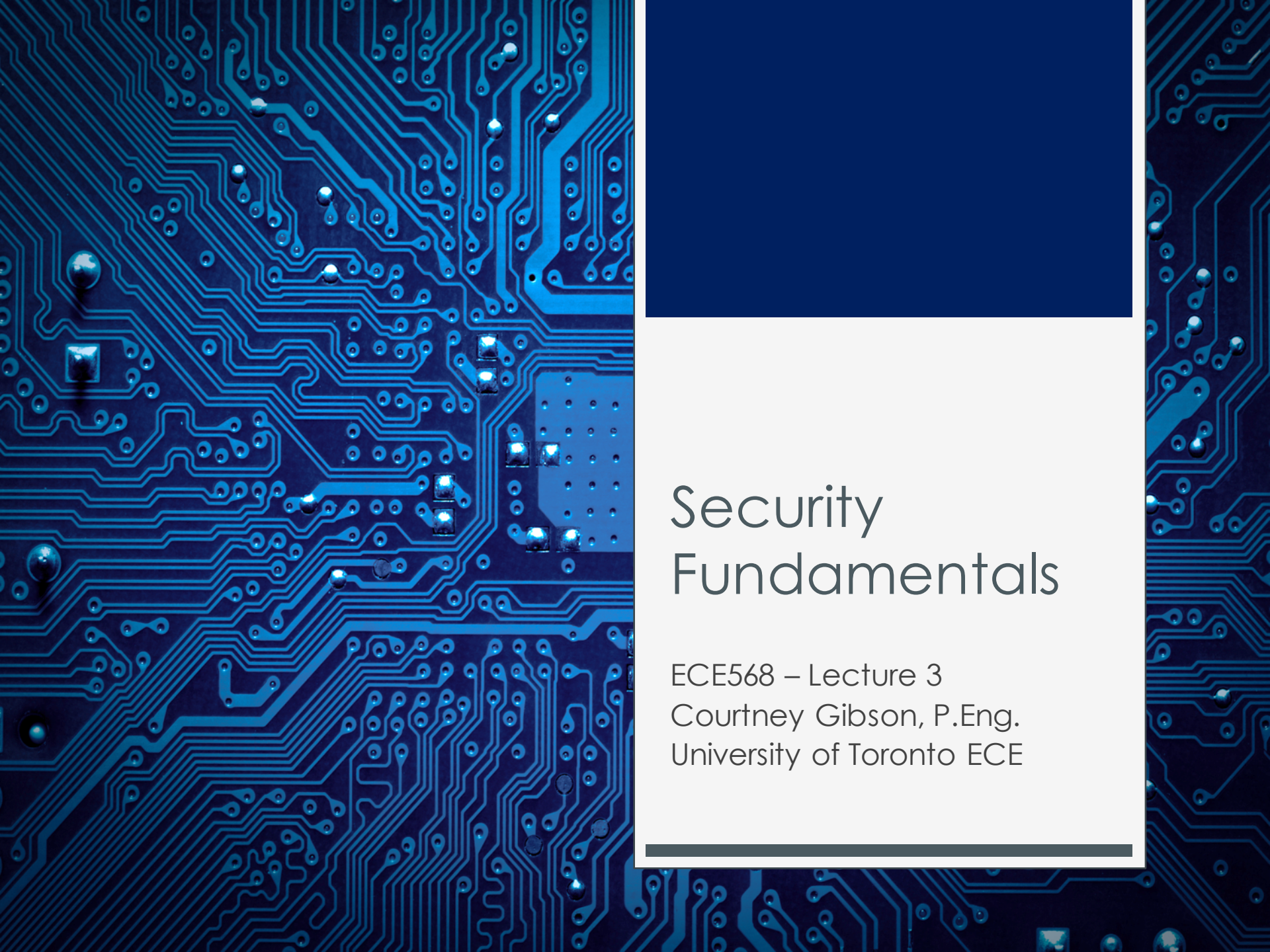


Security Fundamentals

ECE568 – Lecture 3

Courtney Gibson, P.Eng.
University of Toronto ECE

Outline

Security Requirements

- Confidentiality
- Integrity
- Availability

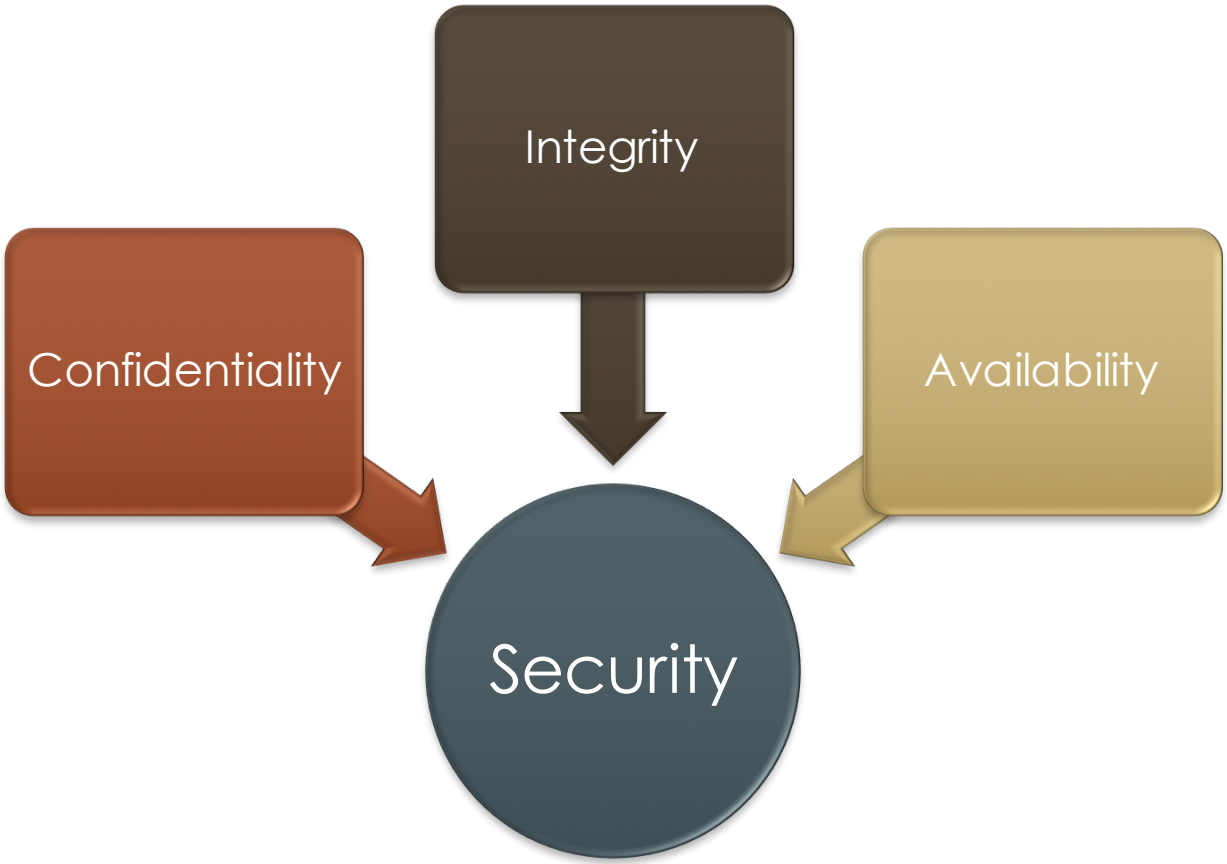
Security Assessment

- What are the threats?
- What is a vulnerability?
- Human issues: User awareness & assurance

Checking Your Assumptions

- Trust

Components of Security



Confidentiality

- **Confidentiality:** the protection of data or resources from exposure.
- There are two aspects of information or resources that are often important to conceal:
  1. The **content** of the data or resource (*e.g.*, passwords, etc.).
  2. The **existence** of the data or resource (as merely knowing that something exists may be damaging).

Integrity

- **Integrity:** the trustworthiness of the data or resource.
- There are two important aspects of a resource or piece of data:
  - Correctness of its **contents**: can we tell if it's been altered?
  - Correctness of its **origin**: can we authenticate who this really came from?

Availability

- **Availability:** ability to access or use the data or resource as desired.
  - A **resource** is available if it is responding to requests
  - **Data** are available if the service that stores the information is up and running
- Availability is generally harder to ensure.

Availability

- A resource could be temporarily unavailable, but at what time does it really become unavailable?
- A system can become unavailable for various reasons (*e.g.*, system crashes, inadequate resources, loss of network connectivity, etc.).
- Systems often deal with availability probabilistically, but for security an active adversary will cause unlikely events to occur more often (*e.g.*, network flooding).

Security: **CIA**

- Any secured system provides Confidentiality, Integrity and Availability to some degree
- However, no system is absolutely secure, and so no system can provide all (any?) of the three absolutely
- Generally, a system is considered secure if the security requirements are met for a given time
  - *e.g.*, confidential data is not leaked until it is declassified

Security Measures

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

Security Measures

Availability is hard to measure quantitatively. Traditionally, it is measured probabilistically, 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 (about 5 minutes/year)

Unfortunately, this approach doesn't apply well to measuring security.

Assessing Security: Threats

A **threat** is any method that can be used to potentially breach security and cause harm.

- When an attacker exercises a threat, it becomes an **attack** (also called an "exploit")
- If the attack is successful, the system is **compromised**

Assessing Security: Threats

Threats can come in many forms, and a good security practitioner learns to identify and assess their seriousness

- *e.g.*, A system is hooked up to the Internet: Internet traffic that is accepted by the system can be a threat.
- *e.g.*, The University allows students to hookup laptops to the wireless network: the laptops pose a threat because they may transport viruses.

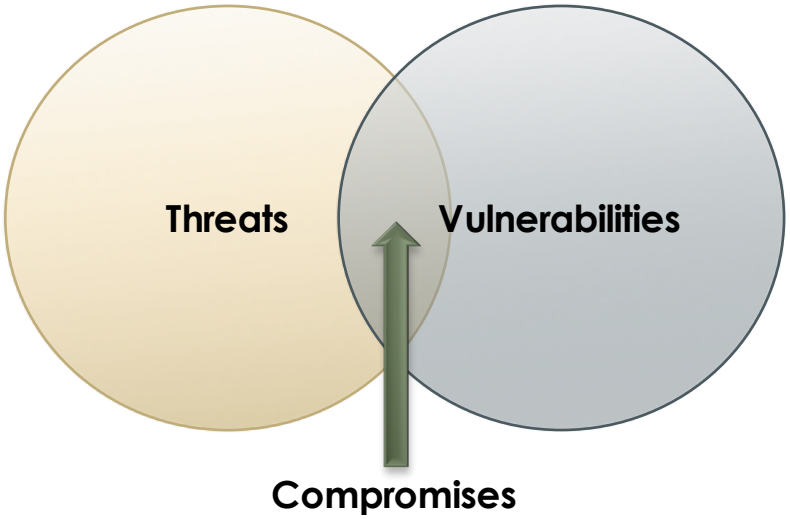
Assessing Security: Vulnerabilities

A **vulnerability** is a flaw that weakens the security of the system. Vulnerabilities are very difficult to identify even after significant amount of testing

- They can be serious:
  - An unchecked string copy allows an attacker to overflow a buffer and execute arbitrary code in a privileged program
  - During configuration, a system administrator forgets to disable debug mode on a program, allowing an attacker to gain privileged information
  - A naïve user does not change the default password on their router from the factory default

Threats, Vulnerabilities and Compromises

Compromises occur when an attacker matches a **threat** (think of these as the attacker's arsenal) with a **vulnerability** (these are weaknesses in the system).



Threats, Vulnerabilities and Compromises

Compromises can lead to data or resource:

- Leak (loss of confidentiality)
- Corruption (loss of integrity)
- Becoming unavailable (loss of availability)

Roughly speaking:

Risk = P(threat) * P(vulnerability) * Cost(compromise)

Human Factors: User Awareness & Assurance

Humans are the leading causes for computer security breaches: they are prone to making mistakes.

Configuration, design, implementation errors:

- Code-reviews, extreme programming
- The amount of checking that has been done to remove errors is defined by the level of **assurance**

Human Factors: User Awareness & Assurance

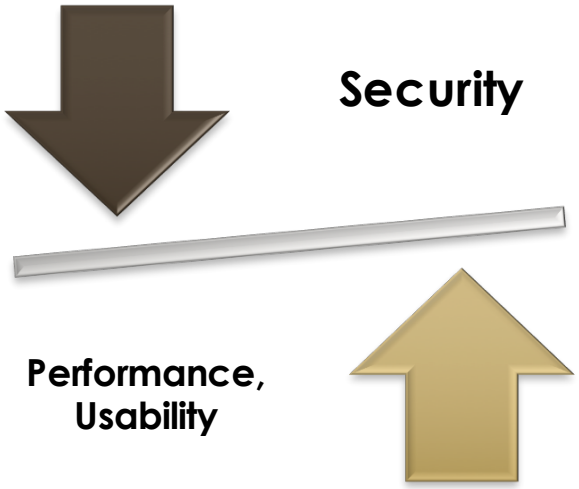
Usage errors:

- Not all people will have an equal understanding of security
- Security-knowledgeable users will create fewer vulnerabilities than unknowledgeable users
- When assessing a system, always keep in mind who created it and who's going to use it

Security Trade-offs

All systems inherently trade off security against performance and usability.

One way of measuring the quality of a security system is how much it impacts the other two.



Trust

Trust defines how much exposure a system has to a particular interface.

- The more a system trusts a component, the more likely that component will be a serious threat, and the more likely that an attacker will find a vulnerability
- The danger is that rather than actively assigning trust, trust in system is usually assigned via the assumptions the designer makes

Case Study: Reflections on Trust

- In 1995, Ken Thompson, one of the original creators of UNIX, was awarded the Turing Award
- His Turing Award lecture is a seminal article on trust called "*Reflections on Trusting Trust*"
- He postulated the ultimate virus that could never be eradicated, as it was able to infect the most trusted portions of a computer system
- What is the most trusted part of a computer system?

Case Study: Reflections on Trust

- Compromise the compiler: whenever it creates a new binary, it adds a virus.
- If it was used to compile another compiler, it would add the replication code to the new compiler binary
- The source code of the original malicious compiler can be discarded!
- What if someone examined the binary?

Case Study: Reflections on Trust

- Ken Thompson's observation was that, if a virus could be hidden in the compiler, it could infect the OS, and all other programs, and use them to hide its existence
- By infecting the highest level of trust, the virus achieves ultimate security (confidentiality, integrity and availability) from the user
- Why?
- Next class we'll discuss exploits that compromise systems by taking advantage of applications that trust their input