CMPSC 497
Security Basics

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Goal of Computer Security

- **Goal**: Prevent information “mishaps”, but don’t stop good things from happening
  - Good things include legal information access for program functionality

- Tradeoff between functionality and security is the key

- E.g, E-Voting
  - Good things: convenience of voting; fast tallying; voting for the disabled; …
  - Good things: fast development, easy process for updates, deploy on lots of platforms, …
  - This convenience creates environment where mishaps may occur
    - Buggy voting software
    - Changed e-voting software by insiders
    - Increased access to adversaries
The Sad Reality

- People are obsessed with providing more functionality
  - Security is secondary
  - Security is an after-thought
    - “We’ll write the software with the required functionality, then our security team will make it secure.”

- Security perspective: integrate security design into the system design process
  - Managing the trade-off between functionality and security from the beginning
The Challenge

- We do not yet have a clear strategy for integrating security into the design process

- **Early Idea**: Formal Assurance
  - Build a formal model of the program
  - Build a formal model of security requirements
  - Prove the program satisfies the security requirements
  - **Challenge**: Implementation may differ from the models
  - Recent success: seL4 microkernel, but only 10K SLOC

- **Current Idea**: Dynamic (Fuzz) Testing
  - Active area of research that we will discuss
E-Voting Application

• Suppose you are building an e-voting application
  ‣ How do you ensure your application satisfies security requirements?

• What does the e-voting application do?
  ‣ Submit a vote (by voter)
  ‣ Store votes
  ‣ Count votes (by tallier)

• What are its security requirements?
  ‣ Let’s see how we reason about security
Risk

- What’s at risk in the e-voting application?
Security Requirements

• Usually security requirements are described in three categories

• Secrecy (Confidentiality)
  ‣ Prevent risk that sensitive data may be leaked to an adversary (e.g., votes)

• Integrity
  ‣ Prevent risk that adversaries may modified data that others depend on (e.g., vote instances and tallies)

• Availability
  ‣ Prevent risk that adversaries block use of critical services (e.g., disable the processing of votes)
Exercise

• Classify each of the following as a violation of confidentiality, of integrity, of availability, or of some combination.
  ‣ Carol changes the amount of Angelo's check from $100 to $1000
  ‣ John copies Mary's homework
  ‣ Eve registers the domain name “psu.edu" and refuses to let Penn State buy or use that domain name.
Adversary

• An adversary is any entity trying to circumvent the security infrastructure
  ‣ The curious and otherwise generally clueless (e.g., script-kiddies)
  ‣ Casual attackers seeking to understand systems
  ‣ Venal people with an ax to grind
  ‣ Malicious groups of largely sophisticated users (e.g, chaos clubs)
  ‣ Competitors (industrial espionage)
  ‣ Governments (seeking to monitor activities)
Threats

- A threat is a specific means by which an adversary can put a system at risk
  - An ability/goal of an adversary (e.g., eavesdrop, fraud, access denial)
  - Independent of what can be compromised
- A threat model is a collection of threats that deemed important for a particular environment
  - A collection of adversary(ies) abilities
  - E.g., a powerful adversary can read and modify all communications and generate messages on a communication channel
Threat Sources

• A challenge is to determine how a program may be threatened by adversaries

• In what ways may an adversary impact CIA?
Threat Sources

• A challenge is to determine how a program may be threatened by adversaries

• In what ways may an adversary impact CIA?

• Adversaries may be able to control the resources used by the program and inputs to the program
  ‣ Obtained via system calls – later researchers described the system calls that may receive adversary-controlled input as a program’s attack surface
Exercise (Threats)

• Identify which of the following are threats to a program.
  ‣ Reading input from arbitrary network (client) IP address
  ‣ Obtaining the PID of a process
  ‣ Reading a file containing client data (e.g., a web page by a web server)
  ‣ Reading a file containing program’s configuration data
  ‣ Opening a file containing program’s configuration data
Vulnerabilities (attack vectors)

• A **vulnerability** is a **flaw** that is **accessible** to an **adversary** who can **exploit** that flaw

• E.g., buffer overflow, file open w/ adversary name

• What is the source of a vulnerability?
  ‣ Bad software (or hardware)
  ‣ Bad design, requirements
  ‣ Bad policy/configuration
  ‣ System Misuse
  ‣ Unintended purpose or environment
  ‣ E.g., student IDs for liquor store
Exercise (Vulnerabilities)

- Identify which of the following that are program vulnerabilities
  - A program flaw (e.g., buffer overflow)
  - A program flaw in writing user input to a program variable
  - A program flaw in writing user input to a program variable that overwrites a function pointer
  - Writing untrusted input to a log file
  - Executing log files
Attacks

- **An attack** occurs when an adversary attempts to **exploit** a vulnerability

- Kinds of attacks
  - Passive (e.g., eavesdropping)
  - Active (e.g., password guessing)
  - Denial of Service (DOS)
    - Distributed DOS – using many endpoints

- A **compromise** occurs when an attack is successful
  - Typically associated with taking over/altering resources
Trust

- To execute a program that obeys security requirements, you must trust something
What Do You Trust?

- To execute a program that obeys security requirements, you must trust something
  - Hardware
  - Operating Systems
  - Libraries
  - Critical input files (configuration)
  - Other programs (system services running with full privilege)
  - Program installers
Trusted Computing Base

- A set of hardware, firmware, and software that are critical to its security
  - Bugs in the TCB may jeopardize the system’s security
  - E.g., a conventional e-voting machine: hardware + system software
- **Components outside of the TCB can misbehave without affecting security**
- In general, a system with a smaller TCB is more trustworthy
- A lot of security research is about how to move components outside of the TCB, or making the TCB smaller
  - E.g., voter-verified paper ballots in E-voting
  - E.g., Proof-Carrying Code removes the compiler from the TCB
Ken Thompson’s Turing Award lecture describes the importance of making the Trusted Computing Base clear.

Do you trust your compiler?

- He describes an approach whereby he can generate a compiler that can insert malicious code on a trigger (e.g., recognizing a login program).

But you can examine the compiler source code

- But, what program compiles the compiler?
- He puts the malicious code in that program.
• **Methodology**: The approach works by generating a malicious binary that is used to compile compilers. Since the compiler source code looks OK and the malice is in the binary compiler compiler, it is difficult to detect.

• **Results**: The system identifies construction of login programs and miscompiles the command to accept a particular password known to the attacker.

• Such a program is an example of a *Trojan horse* malware
Turtles all the way down ...

• **Take away**: Thompson states the “obvious” moral that “you cannot trust code that you did not totally create yourself.” We all depend on code, but constructing a basis for trusting it is very hard, even today.

• ... or “**trust in security is an infinite regression ...**”

“A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise." The scientist gave a superior smile before replying, "What is the tortoise standing on?" "You're very clever, young man, very clever", said the old lady. "But it's turtles all the way down!"

Trust

- **Trust** refers to the degree to which a principal is expected to behave
  - What the principal not expected to do?
    - E.g., not expose password
  - What the principal is expected to do (obligations)?
    - E.g., obtain permission, refresh

- A **trust model** describes, for a particular environment, who is trusted to do what?

- Note: you make trust decisions every day
  - Q: What are they?
  - Q: Whom do you trust?
Security Model

• A security model is the combination of a trust and threat models that address the set of perceived risks
  ‣ The “security requirements” used to develop some cogent and comprehensive design
  ‣ Every design must have security model
    • LAN network or global information system
    • Java applet or operating system
• This class is going to talk a lot about security models
  ‣ What are the security concerns (risks)?
  ‣ Who are our adversaries?
  ‣ What are the threats?
  ‣ Who do we trust and to do what?
• Systems must be explicit to be secure.
Security Model - E-voting

• Who are the principals?
  ‣ Voters, Admins, Talliers, Others

• Who are adversaries?

• Which commands may be threatened (attack surface)?
  ‣ Start Program
  ‣ Submit a vote (by voter)
  ‣ Count votes (by tallier)

• Who must the application trust? To do what?
  ‣ Of principals above
Security Model - E-voting

• Security model is common in all security research
  ‣ To establish boundary between entities that will follow security requirements (trust) and those that may not (threats)

• What are the security requirements for e-voting?
  ‣ Secrecy, Integrity, and Availability

• The security problem is to define mechanisms and policies to enable the program and trusted entities to prevent threats from leading to violations of security requirements under the security model
Security Mechanisms

• What mechanisms and policies are necessary to enforce
  ‣ Secrecy, Integrity, and Availability

• Authentication
  ‣ Determine which principal submits a command

• Authorization
  ‣ Enforce control on access to vote objects

• Challenge: often need to extend program to control access to its objects – currently done in an ad hoc way
Take Away

• Our goal is to protect the confidentiality, integrity, and availability of our programs and their resources
  ‣ Without impacting the program functionality

• What resources? Ones that present risks

• What threats? From untrusted principals accessing the program’s attack surface

• How can we satisfy security? Program (and entities program trusts) can enforce security requirements using security mechanisms