OREGON PROGRAMMING LANGUAGES SUMMER SCHOOL AT BOSTON UNIVERSITY

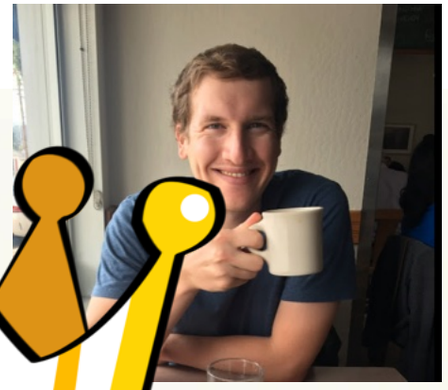
Harvard John A. Paulson School of Engineering and Applied Sciences

Language-Based Security

Stephen Chong, Harvard University


Discovery
SHARK
WEEK


STEVE WEEK
STEVE WEEK

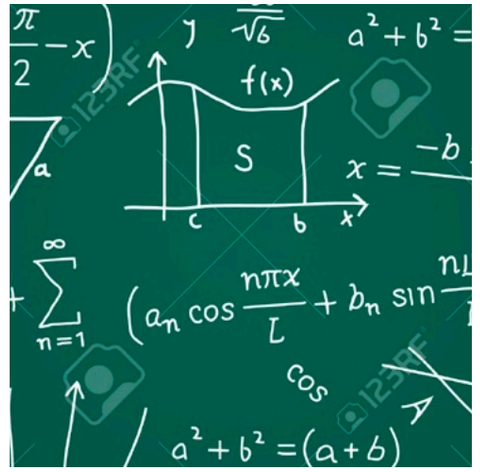


STEVE'S
& ALLEYS

Formal Methods for Security

- Modeling computer systems to understand and enforce security guarantees

Model of computer system



Reason about model

E.g., specifying security, proving the system satisfies (or violates) security, ...

E.g., enforcement mechanisms, defense mechanisms, ...

Real computer system

Formal Methods

- Approaches to reasoning about computational entities
  - Logical or mathematical descriptions
  - Enable drawing reliable conclusions about behavior
- Enable modeling, verifying, and synthesizing computer systems.
- Can be usefully applied with varying degrees of rigor

Security

- Often described as "making sure bad things don't happen" in computer systems
- Distinct from *functional correctness*
  - i.e., making sure the system does the right thing when provided with appropriate input
- Many aspects to it
  - Authentication, authority, confidentiality, integrity, availability, abstraction violation, non-repudiation, ...
- We will examine some of these later

Advantages of Formal Methods

- Clear and explicit statement of what we mean by *security*
- Modeling often requires explicating threat model, assumptions, goals, ...
  - Lead to clarity and insight
- Formal methods can lead to **proofs** of security
  - Up to the assumptions of the model, computer system guaranteed to be secure!
  - Against entire classes of attacks!
  - Even against as-yet-unknown attacks!
  - Contrast with the "patch latest vulnerability" mentality

Disadvantages of Formal Methods

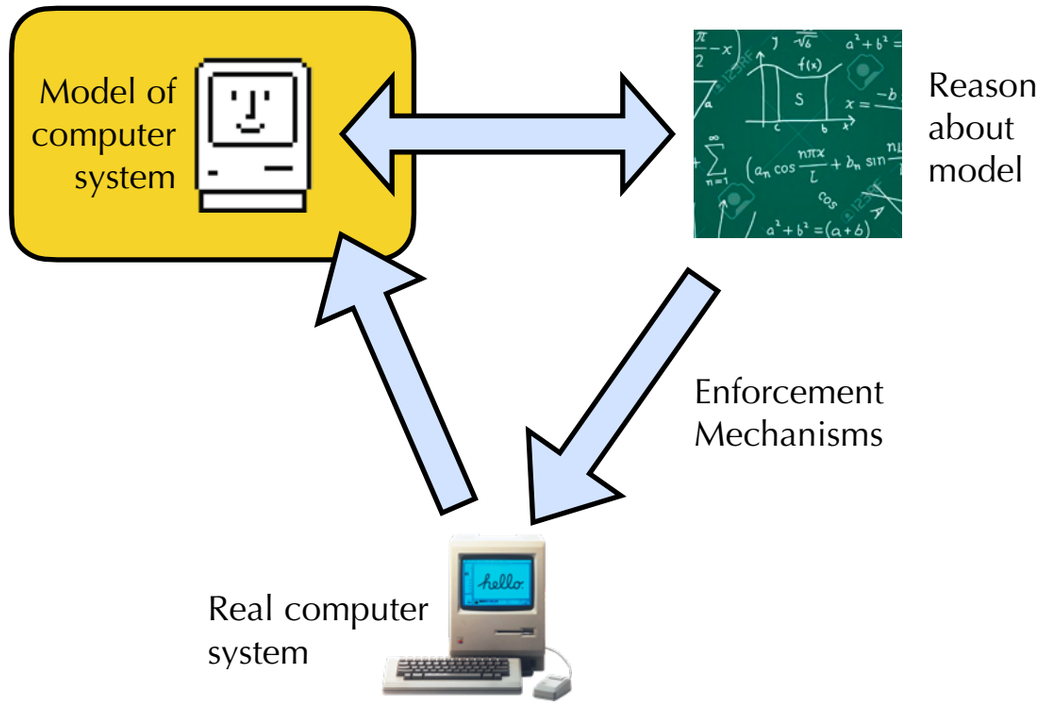
- Security guarantees only as good as model
- Precisely phrasing a security guarantee can lead to difficult/unintuitive/opaque security guarantees
- Applying formal methods can be expensive
  - Difficult, requires significant expertise, time-consuming, ...

Language-Based Security

- Use of **Programming Language concepts and techniques** to reason about and enforce security
- Great fit for formal methods for security!

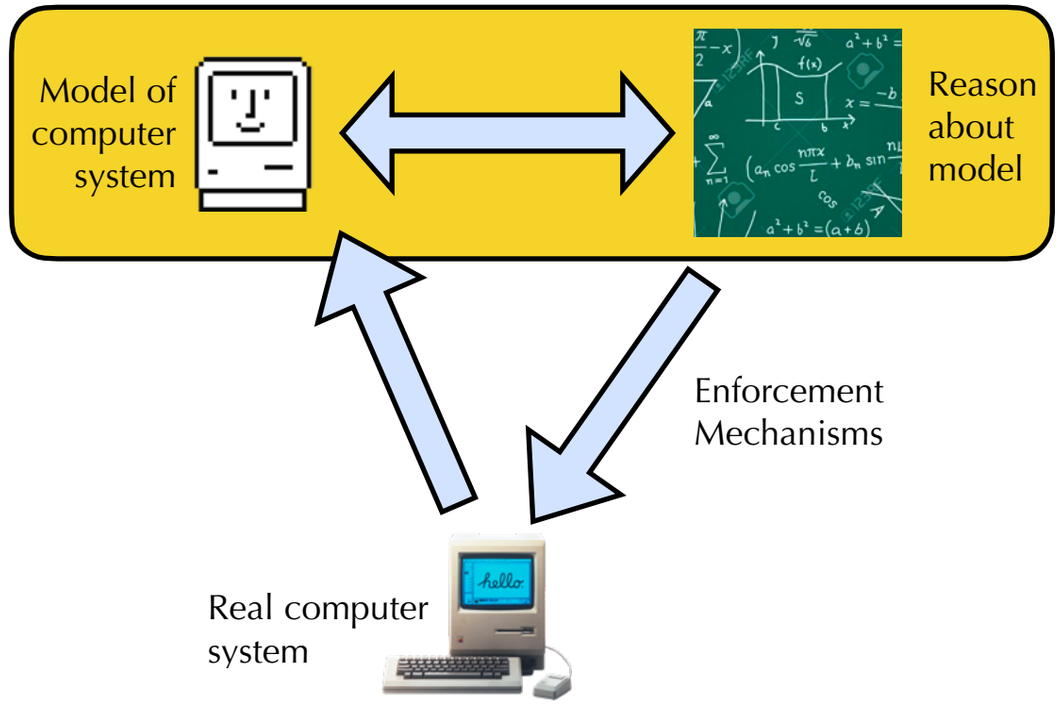
Language-Based Reasoning

- PL has rich tradition of simple formal models of languages
  - Lambda calculus, Pi calculus, Imperative calculi, labeled transition systems, ...
- Also often good models for computer systems



Language-Based Reasoning

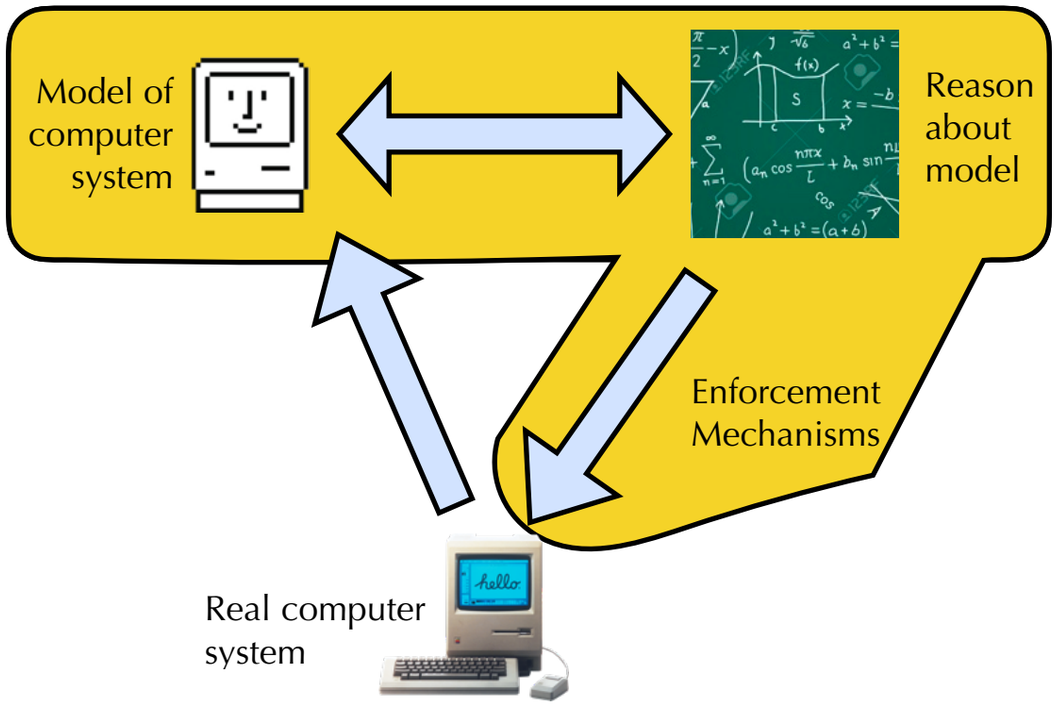
- PL has rich tradition of simple formal models of languages
  - Lambda calculus, Pi calculus, Imperative calculi, labeled transition systems, ...



- Also often good models for computer systems
- Techniques for reasoning and proofs in calculi translate to techniques for reasoning and proofs about security

PL Enforcement Techniques

- Enforcement mechanisms in PL-based models often translate to useful real mechanisms
  - Because we use PL to implement systems
- E.g., type systems, reference monitors, static analyses, …



Abstraction Enforcement

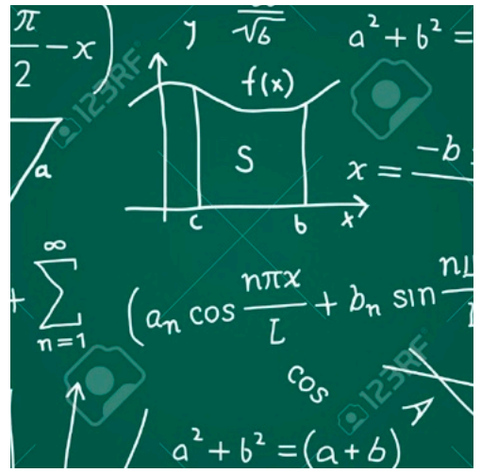
- Aspects of security of system relies on correctness/security of application code
- But if system execution doesn't obey application code, hard to reason correctly about security
  - E.g., buffer overflow errors injecting arbitrary computation
  - E.g., linking to arbitrary binary libraries

Techniques to enforce language semantics/ abstractions $\subseteq$ Language-based Security

Case Study: Noninterference

- Setting: a computer system that handles information of varying sensitivity
  - E.g., Military Multi-Level Security (MLS)
    - Data labeled e.g., "Nuclear Confidential", "Signals Top Secret", "Crypto Unclassified"
  - E.g., Web app with different users
- Key security idea: a user/adversary should not learn information inappropriately
- (We will be examining this in much more depth later)

Model of computer system

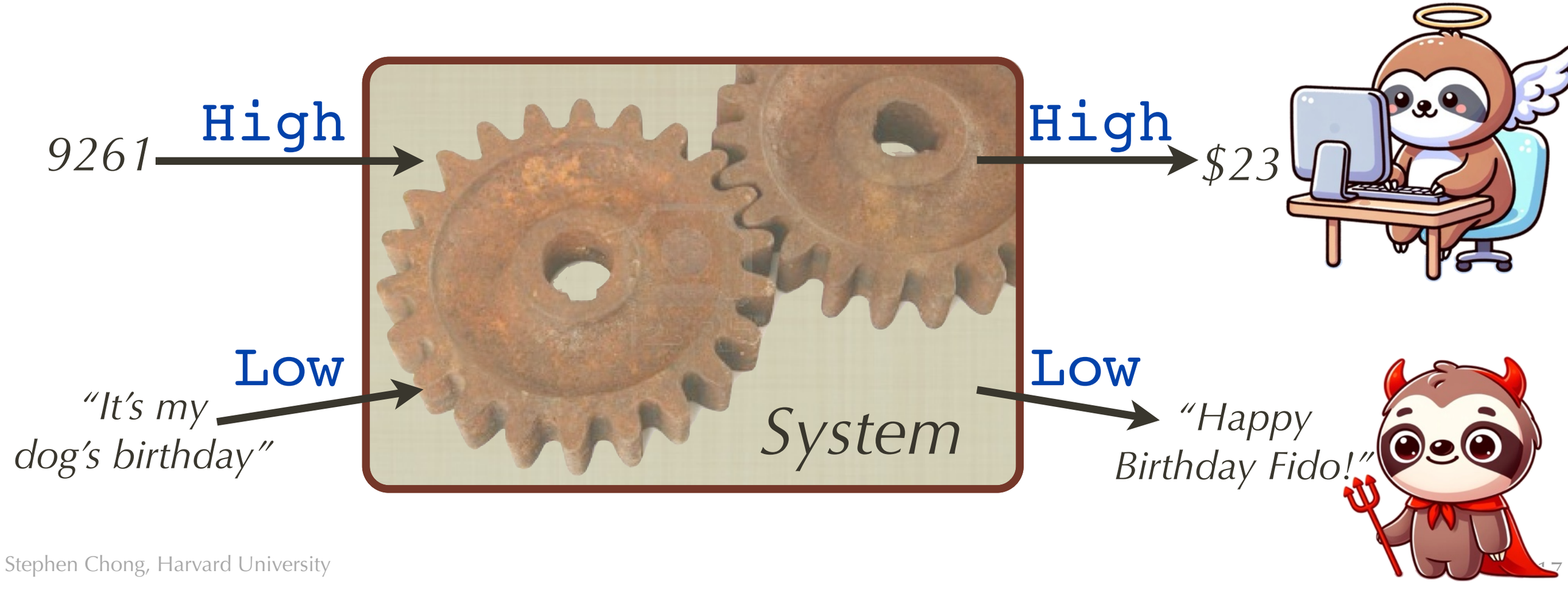


Reason about model

Real computer system

Input, Output, and Observation

- Let's consider a simplified setting, just two security levels, Low and High
  - Think "public information" for Low and "secret information" for High
- Key idea, diagrammatically:



A Simple Model

- Let's model the computer system with a simple imperative language (IMP)

arithmetic expressions	$a \in \mathbf{Aexp}$	$a ::= x \mid n \mid a_1 + a_2 \mid a_1 \times a_2$
boolean expressions	$b \in \mathbf{Bexp}$	$b ::= \mathbf{true} \mid \mathbf{false} \mid a_1 < a_2$
commands	$c \in \mathbf{Com}$	$c ::= \mathbf{skip} \mid x := a \mid c_1; c_2$
		$\mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2$
		$\mid \mathbf{while}\ b\ \mathbf{do}\ c$

IMP Semantics

- Store σ is a function from variables to ints

$$\Downarrow_{\mathbf{Aexp}} \subseteq \mathbf{Aexp} \times \mathbf{Store} \times \mathbf{Int}$$

$$\Downarrow_{\mathbf{Bexp}} \subseteq \mathbf{Bexp} \times \mathbf{Store} \times \mathbf{Bool}$$

$$\Downarrow_{\mathbf{Com}} \subseteq \mathbf{Com} \times \mathbf{Store} \times \mathbf{Store}$$

Arithmetic expressions.

$$\frac{}{\langle n, \sigma \rangle \Downarrow n} \qquad \frac{}{\langle x, \sigma \rangle \Downarrow n} \text{ where } \sigma(x) = n$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 + a_2, \sigma \rangle \Downarrow n} \text{ where } n = n_1 + n_2 \qquad \frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 \times a_2, \sigma \rangle \Downarrow n} \text{ where } n = n_1 \times n_2$$

IMP Semantics

Boolean expressions.

$$\frac{}{\langle \textbf{true}, \sigma \rangle \Downarrow \textbf{true}} \qquad \frac{}{\langle \textbf{false}, \sigma \rangle \Downarrow \textbf{false}}$$

$$\frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 < a_2, \sigma \rangle \Downarrow \textbf{true}} \text{ where } n_1 < n_2 \qquad \frac{\langle a_1, \sigma \rangle \Downarrow n_1 \qquad \langle a_2, \sigma \rangle \Downarrow n_2}{\langle a_1 < a_2, \sigma \rangle \Downarrow \textbf{false}} \text{ where } n_1 \geq n_2$$

Commands.

$$\textsc{Skip}\ \frac{}{\langle \textbf{skip}, \sigma \rangle \Downarrow \sigma} \qquad \textsc{Asg}\ \frac{\langle a, \sigma \rangle \Downarrow n}{\langle x := a, \sigma \rangle \Downarrow \sigma[x \mapsto n]} \qquad \textsc{Seq}\ \frac{\langle c_1, \sigma \rangle \Downarrow \sigma' \qquad \langle c_2, \sigma' \rangle \Downarrow \sigma''}{\langle c_1; c_2, \sigma \rangle \Downarrow \sigma''}$$

$$\textsc{If-T}\ \frac{\langle b, \sigma \rangle \Downarrow \textbf{true} \qquad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle \Downarrow \sigma'} \qquad \textsc{If-F}\ \frac{\langle b, \sigma \rangle \Downarrow \textbf{false} \qquad \langle c_2, \sigma \rangle \Downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle \Downarrow \sigma'}$$

$$\textsc{While-F}\ \frac{\langle b, \sigma \rangle \Downarrow \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \Downarrow \sigma} \qquad \textsc{While-T}\ \frac{\langle b, \sigma \rangle \Downarrow \textbf{true} \qquad \langle c, \sigma \rangle \Downarrow \sigma' \qquad \langle \textbf{while } b \textbf{ do } c, \sigma' \rangle \Downarrow \sigma''}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \Downarrow \sigma''}$$

Security Condition

- Inputs are the store before the program executes, outputs are the final store
- Let's designate a subset of the variables as "Low", meaning the attacker can observe them, and the rest as "High"
- Context Γ will be a function from variables to {Low, High}
- Write $\sigma_1 =_{\text{Low}} \sigma_2$ if states σ_1 and σ_2 are equal on all low variables
  - For all x, if $\Gamma(x) = \text{Low}$ then $\sigma_1(x) = \sigma_2(x)$
- Definition: Program c is **noninterfering** if:

  For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2$,

  if $\sigma_1 =_{\text{Low}} \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$

  then $\sigma'_1 =_{\text{Low}} \sigma'_2$

Examples

Γ(x)=High
Γ(y)=Low

```
y := x + 42;
```

```
x := x + y;
y := y + 42;
```

```
if (x > 0)
   y := 1
else
   y := 0
```

```
y := 0;
if (x > 0)
   y := 1
```

Examples

Γ(x)=High
Γ(y)=Low

```
while (x > 0) do skip
```

```
y := 0;
while (x > 0) do y := 1
```

```
y := encrypt(x)
```

Road Map

- Intro
  - Formal Methods for Security
  - Language-Based Security
  - Case Study: Noninterference
- Primer on Computer Security
- Information Flow
  - Semantics
  - Enforcement
  - Beyond confidentiality
- Enforcing Language Abstractions

Road Map

- Intro **Mon**
  - Formal Methods for Security
  - Language-Based Security
  - Case Study: Noninterference
- Primer on Computer Security
- Information Flow **Tue**
  - Semantics
  - Enforcement **Wed**
  - Beyond confidentiality
- Enforcing Language Abstractions **Thu**

Road Map

- Intro
  - Formal Methods for Security
  - Language-Based Security
  - Case Study: Noninterference
- **Primer on Computer Security**
- Information Flow
  - Semantics
  - Enforcement
  - Beyond confidentiality
- Enforcing Language Abstractions

OREGON PROGRAMMING LANGUAGES SUMMER SCHOOL AT BOSTON UNIVERSITY

Harvard John A. Paulson School of Engineering and Applied Sciences

Primer on Computer Security

- Some of the key concepts in computer security
- Some of which may crop up in future lectures

Primer Outline

- Policy
- Enforcement
- Trust
- Threat Model
- Some good security principles

Policy vs Enforcement

- **Policy** is (or leads to) the **security guarantee** we want to achieve
- **Enforcement** is how we achieve the security guarantee
  - Sometime called the *mechanism*
- **Threat model** (aka adversary model, attacker model): assumptions about the abilities and/or motivations of the attacker
- **Trusted Computing Base** (TCB): components of the system that are trusted
- Goal: given the assumptions of the threat model and assuming components in the TCB work correctly, the enforcement mechanism should ensure the desired security guarantee.

Policy

- "A security policy is a statement of what is, and what is not, allowed"
  - [Bishop]
- "Security policies legislate behavior by people, computers, executing programs, communications channels, and other system entities capable of taking action."
  - [Schneider]
  - Refer to such entities as **principals**

Information Security

- Foundation of computer security is **CIA**
  - Confidentiality
  - Integrity
  - Availability

Confidentiality

- The concealment of information or resources
- Access control mechanisms support confidentiality
  - E.g., cryptography, discretionary access control, …
- But may be concerned not just with restricting **access** to information, but also with restricting **propagation** of and **knowledge** of information.
- Note: sometimes the **existence** of information itself is confidential
  - E.g., What resources a client is using is confidential, but so is the existence of the client
  - E.g., Contents of message from employee to HR is confidential, but so is the fact that a message was sent at all

Integrity

- Trustworthiness of data or resources
  - Often about preventing improper or unauthorized changes
- Includes:
  - Data integrity (i.e. content of information)
    - Contamination, supression
  - Origin integrity (aka authentication)
- Mechanisms to support integrity typically either **prevention** or **detection**
  - Prevention: prevent unauthorized changes to data
  - Detection: determine when data/resources no longer trustworthy
- In many settings, integrity is dual to confidentiality
  - Cryptography, information flow, ...

Availability

- Ability to use information or resource as desired
- Attacks to block availability often called **denial of service attacks**

Privacy

- Related to the confidentiality of personal information
- May also include right of individuals to control or influence what info about them is collected, stored, shared
- Often broader societal question
  - Many laws and regulations

Non-Repudiation

- A party that took an action is unable to plausibly deny that they did so
- E.g.,
  - After signing a document, can not deny that they did so
  - After sending a message, can not deny that the message was sent by them
- Related to integrity

Example: Electronic Voting

- Desirable properties for electronic voting
- **Verifiability**: The final tally is verifiably correct
  - **Voter/individual verifiability**: Each voter can check that their own vote is included in the tally
  - **Universal verifiability**: Anyone can check that all votes cast are counted, that only authorized votes are counted, and that no votes are changed during counting
- **Coercion Resistance**: Voters cannot prove whether or how they voted
  - Might involve ability to repudiate!
- **Availability**: the voting system should be available to voters during the voting period

Example: Noninterference

- Here is the security condition from earlier:
  - Context Γ is function from variables to {Low, High}
  - Write $\sigma_1 =_{\text{Low}} \sigma_2$ if states σ_1 and σ_2 are equal on all low variables: For all x, if $\Gamma(x) = \text{Low}$ then $\sigma_1(x) = \sigma_2(x)$
  - Definition: Program c is **noninterfering** if:

    For all σ_1, σ_2, σ'_1, σ'_2,

    if $\sigma_1 =_{\text{Low}} \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$

    then $\sigma'_1 =_{\text{Low}} \sigma'_2$
- What is and isn't allowed here? What are we trying to achieve?

Primer Outline

- Policy
- Enforcement
- Trust
- Threat Model
- Some good security principles

Enforcement

- Mechanism by which we ensure a system satisfies a security policy
- Many kinds of mechanisms
  - Cryptography
  - Type systems
  - Program Analyses
  - Program Rewriting, Runtime Monitors, Inlined Reference Monitors
  - APIs and frameworks
    - E.g. access control frameworks
  - Language abstractions
    - E.g., information hiding
  - Patterns/idioms
    - E.g., escaping SQL strings
  - Intrusion Detection Systems
  - Firewalls
  - ...

What is a good mechanism?

- Correct (i.e., actually achieves the desired security policy)
- Hard or impossible to get wrong
  - Examples for and against?
- Can not be subverted or bypassed, aka **complete mediation**
  - By attacker, developer, etc.
- Easy to use (possibly even transparent to developer)
- Separates policy and mechanism
  - i.e., what we want to achieve from how we achieve it
- Useful
  - Most secure system is one that doesn't turn on...
- These may be in conflict with each other
  - E.g., to be "easy to use", a developer may want to disable a mechanism locally if the enforcement mechanism is too strict
    - Like `unsafe` in Rust
    - cf. declassification for noninterference

The "Gold Standard"

- Authorization
- Authentication
- Audit

Au comes from *aurum*,
the Latin name for gold.

- Many aspects of enforcement rely on one or more of these

Authentication

- Confirm identity of entity
- Based on one or more of:
  - Something you know
    - E.g., password, secret
  - Something you have
    - Mobile phone, smart card, hardware authentication device
    - Physical key
  - Something you are
    - Biometric, e.g., fingerprint, iris pattern

Authorization

- Can requested actions proceed?
  - Is the requestor authorized to do the action?
- May rely on authentication
- Common mechanisms include:
  - Access control mechanisms
    - E.g., Role-based access control
  - Capabilities
  - Authorization logic

Audit

- Records system activity, attributing actions to some responsible principal
- Supports **accountability**: holding people (legally) responsible for their actions
  - Might help enforce security policy by
    - (1) providing incentive for people to follow policy; and/or
    - (2) retroactively enforcing policy

Example Authorization Mechanism: Discretionary Access Control

- Discretionary = up to the owner to decide access rights
  - E.g., owner of a file can set the permissions for the file
  - By contrast, with Mandatory Access Control, the system determines access rights
- If we are using access control to protect actions (e.g., launching of missiles, opening door), may be a good mechanism for the policy
  - Policy (access control lists) separate from how the policy is enforced
- If we are using access control to enforce confidentiality, may not be a good mechanism
  - Actual goal is preventing some entities from learning information
  - Different from whether they can open a file

Primer Outline

- Policy
- Enforcement
- Trust
- Threat Model
- Some good security principles

Trust

- Trust is inevitable. Need to trust something
- Ideally, want to trust as little as possible.
  - You can only be hurt by those you trust!



"Draw a picture that conveys the message you can only be hurt by those you trust"

Trusted vs Trustworthy

- **Trustworthy**: worthy of trust
  - Exhibits all and only expected functionality/behavior, and is some compelling reason to believe that is the case
- **Trusted** is not the same as trustworthy
  - Trust may be misplaced
- A component is **trusted** if its failure can violate the security goal
- **Trusted Computing Base** is the set of components that are trusted
  - Sometimes trustworthy components are not included in TCB, i.e., TCB is only the trusted components that might fail

Threat Model

- Aka Attacker Model, Adversary Model, …
- Assumptions about the ability of an adversary
- Good to be clear and explicit about assumptions
  - Help determine whether a mechanism does or does not achieve desired security
- E.g., what can the attacker do with respect to communication channels?
  - Observe that messages are sent? Observe length of messages? Contents of messages? Send messages itself?
- E.g., does the attacker have physical access to the computer?
  - May be able to snoop on the memory bus to see what accesses are being made
  - May be able to more easily cause random bitflips
  - May be able to turn off power
  - May be able to connect devices (e.g., USB keys)
  - May be able to repeatedly try attacks without observation (e.g., smartcard)

Threat Model

- E.g., can the attacker execute new programs/ code? If so, is the attacker restricted to well-typed programs?
- E.g., is the attacker computationally limited?
  - An omniscient attacker can break cryptography…
- E.g., is the attacker able to encrypt additional messages using the secret key?

Security Up To Threat Model

- Recall our goal: *given the assumptions of the threat model and assuming components in the TCB work correctly, the enforcement mechanism should ensure the desired security guarantee.*
- If threat model assumptions incorrect, system may not be secure
  - Generally good to assume stronger adversary
- Evaluating whether assumptions are reasonable are typically beyond scope of formal methods
- But making threat model explicit helps clarify what assumptions are being made

Attack Surface

- Points where attacker can interact with system
- Smaller the attack surface, fewer opportunities for attacks
- Attack surface tightly connected with threat model
- Isolating systems (or components) as much as possible can reduce attack surface

Primer Outline

- Policy
- Enforcement
- Trust
- Threat Model
- Some good security principles

Some Good Security Principles

- Defense in depth
  - Have multiple mechanisms to defend against attacks
- Principle of Least Privilege
  - Principals should be given only the minimum privileges/authority needed to accomplish its task
- Privilege Separation
  - Different actions should require different privileges
  - Supports principle of least privilege
  - But may be difficult to manage huge numbers of privileges
- Security by obscurity is generally bad
  - i.e., security of system relying on keeping mechanism design secret is not a good approach
  - Yes, hiding design information makes attacking more difficult
  - But the design often leaks eventually; more scrutiny of design improves it

References

- "Computer Security: Art and Science" by Matt Bishop
- Draft chapters for an as-yet untititled textbook on cybersecurity, by Fred Schneider
  - https://www.cs.cornell.edu/fbs/fullist.html