Principles for secure design

Some of the slides and content are from Mike Hicks’ Coursera course
Making secure software
Making secure software

• **Flawed approach**: Design and build software, and *ignore security at first*
  • Add security once the functional requirements are satisfied
Making secure software

• **Flawed approach**: Design and build software, and *ignore security at first*
  • Add security once the functional requirements are satisfied

• **Better approach**: *Build security in* from the start
  • Incorporate security-minded thinking into all phases of the development process
Development process

• Many development processes; **four common phases**:  
  - Requirements
  - Design
  - Implementation
  - Testing/assurance

• Where does **security engineering** fit in?  
  - All phases!
Security engineering

Phases

• Requirements
• Design
• Implementation
• Testing/assurance

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

**Phases**

- Requirements
- Design
- Implementation
- Testing/assurance

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

Phases

- Requirements
- Design
- Implementation
- Testing/assurance

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

**Phases**
- Requirements
- Design
- Implementation
- Testing/assurance

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

Phases

- Requirements
- Design
- Implementation
- Testing/assurance

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

Phases

• Requirements
• Design
• Implementation
• Testing/assurance

Activities

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

Phases

• Requirements
• Design
• Implementation
• Testing/assurance

Activities

Security Requirements
Abuse Cases
Architectural Risk Analysis
Security-oriented Design
Code Review (with tools)

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Security engineering

**Phases**

- Requirements
- Design
- Implementation
- Testing/assurance

**Activities**

- Security Requirements
- Abuse Cases
- Architectural Risk Analysis
- Security-oriented Design
- Code Review (with tools)
- Risk-based Security Tests

Note that different SD processes have different phases and artifacts, but all involve the basics above. We’ll keep it simple and refer to these.
Security engineering

Phases

- Requirements
- Design
- Implementation
- Testing/assurance

Activities

- Security Requirements
- Abuse Cases
- Architectural Risk Analysis
- Security-oriented Design
- Code Review (with tools)
- Risk-based Security Tests
- Penetration Testing

Note that different SD processes have different phases and artifacts, but all involve the basics above. We'll keep it simple and refer to these.
Designing secure systems

• **Model** your threats

• Define your **security requirements**
  • What distinguishes a security requirement from a typical “software feature”?

• Apply good security **design principles**
Threat Modeling
Threat Model
Threat Model

• The threat model makes explicit the adversary’s assumed powers
  • Consequence: The threat model must match reality, otherwise the risk analysis of the system will be wrong
Threat Model

• The **threat model** makes explicit the adversary’s **assumed powers**
  • Consequence: The threat model must match reality, otherwise the risk analysis of the system will be wrong

• The threat model is **critically important**
  • If you are not explicit about what the attacker can do, how can you assess whether your design will repel that attacker?
A few different network threat models

Malicious user
A few different network threat models

Malicious user

Co-located user
A few different network threat models

- Malicious user
- Co-located user
- Compromised server
A few different network threat models

- Malicious user
- Snooping
- Co-located user
- Compromised server
Threat-driven Design
Threat-driven Design

- Different threat models will elicit different responses
Threat-driven Design

- Different threat models will elicit different responses

- **Only malicious users:** implies message traffic **is safe**
  - No need to encrypt communications
  - This is what telnet remote login software assumed
Threat-driven Design

• Different threat models will elicit different responses

• **Only malicious users:** implies *message* traffic is safe
  • No need to encrypt communications
  • This is what `telnet` remote login software assumed

• **Snooping attackers:** means *message* traffic is visible
  • So use encrypted wifi (link layer), encrypted network layer (IPsec), or encrypted application layer (SSL)
    - Which is most appropriate for your system?
 Threat-driven Design

• Different threat models will elicit different responses

• **Only malicious users:** implies message traffic is safe
  • No need to encrypt communications
  • This is what telnet remote login software assumed

• **Snooping attackers:** means message traffic is visible
  • So use encrypted wifi (link layer), encrypted network layer (IPsec), or encrypted application layer (SSL)
    - Which is most appropriate for your system?

• **Co-located attacker:** can access local files, memory
  • Cannot store unencrypted secrets, like passwords
  • Likewise with a compromised server
Bad Model = Bad Security

• Any **assumptions** you make in your model are potential **holes that the adversary can exploit**
Bad Model = Bad Security

• Any **assumptions** you make in your model are potential **holes that the adversary can exploit**

• E.g.: Assuming no snooping users **no longer valid**
  - *Prevalence of wi-fi networks in most deployments*
Bad Model = Bad Security

• Any **assumptions** you make in your model are potential **holes that the adversary can exploit**

• E.g.: Assuming no snooping users **no longer valid**
  • *Prevalence of wi-fi networks in most deployments*

• Other mistaken assumptions
  • **Assumption**: Encrypted traffic carries no information
    - Not true! By analyzing the size and distribution of messages, you can infer application state
Bad Model = Bad Security

• Any **assumptions** you make in your model are potential **holes that the adversary can exploit**

• E.g.: Assuming no snooping users **no longer valid**
  • *Prevalence of wi-fi networks in most deployments*

• Other mistaken assumptions
  • **Assumption:** Encrypted traffic carries no information
    - Not true! By analyzing the size and distribution of messages, you can infer application state
  • **Assumption:** Timing channels carry little information
    - Not true! Timing measurements of previous RSA implementations could be used eventually reveal a remote SSL secret key
Finding a good model
Finding a good model

- Compare against similar systems
  - What attacks does their design contend with?
Finding a good model

• **Compare against similar systems**
  • What attacks does their design contend with?

• **Understand past attacks and attack patterns**
  • How do they apply to your system?
Finding a good model

• **Compare against similar systems**
  • What attacks does their design contend with?

• **Understand past attacks and attack patterns**
  • How do they apply to your system?

• **Challenge assumptions in your design**
  • What happens if an assumption is untrue?
    - What would a breach potentially cost you?
  • How hard would it be to get rid of an assumption, allowing for a stronger adversary?
    - What would that development cost?
Security Requirements
Security Requirements
Security Requirements

• **Software requirements** typically about **what** the software should do
Security Requirements

- **Software requirements** typically about **what** the software should do

- We also want to have **security requirements**
Security Requirements

• **Software requirements** typically about **what** the software should do

• We also want to have **security requirements**
  - **Security-related goals** (or **policies**)
    - **Example:** One user’s bank account balance should not be learned by, or modified by, another user, unless authorized
Security Requirements

• **Software requirements** typically about **what** the software should do

• We also want to have **security requirements**
  - **Security-related goals** (or **policies**)
    - **Example**: One user's bank account balance should not be learned by, or modified by, another user, unless authorized
  - **Required mechanisms** for enforcing them
Security Requirements

• **Software requirements** typically about what the software should do

• We also want to have **security requirements**
  - **Security-related goals** (or **policies**)
    - **Example**: One user's bank account balance should not be learned by, or modified by, another user, unless authorized
  - **Required mechanisms** for enforcing them
    - **Example**:
      1. Users identify themselves using passwords,
Security Requirements

• **Software requirements** typically about what the software should do

• We also want to have **security requirements**
  - **Security-related goals** (or **policies**)
    - **Example**: One user’s bank account balance should not be learned by, or modified by, another user, unless authorized
  - **Required mechanisms for enforcing them**
    - **Example**:
      1. Users identify themselves using passwords,
      2. Passwords must be “strong,” and
Security Requirements

• **Software requirements** typically about what the software should do

• We also want to have **security requirements**
  - **Security-related goals** (or **policies**)
    - **Example:** One user's bank account balance should not be learned by, or modified by, another user, unless authorized
  - **Required mechanisms** for enforcing them
    - **Example:**
      1. Users identify themselves using passwords,
      2. Passwords must be “strong,” and
      3. The password database is only accessible to login program.
Typical *Kinds* of Requirements
Typical *Kinds* of Requirements

- Policies
  - Confidentiality (and Privacy and Anonymity)
  - Integrity
  - Availability
Typical *Kinds* of Requirements

- **Policies**
  - Confidentiality (and Privacy and Anonymity)
  - Integrity
  - Availability

- Supporting **mechanisms**
  - Authentication
  - Authorization
  - Audit-ability
Supporting mechanisms

These relate identities (“principals”) to actions

Authentication  Authorization  Audit-ability
Supporting mechanisms

These relate identities ("principals") to actions

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Authorization</th>
<th>Audit-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can a system tell who a user is</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Supporting mechanisms

These relate identities ("principals") to actions

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Authorization</th>
<th>Audit-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can a system tell who a user is</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What we know

What we have

What we are

>1 of the above =

*Mult-factor authentication*
Supporting mechanisms

These relate identities ("principals") to actions

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Authorization</th>
<th>Audit-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can a system tell <em>who a user is</em></td>
<td>How can a system tell <em>what a user is allowed</em> to do</td>
<td></td>
</tr>
</tbody>
</table>

What we know
What we have
What we *are*
>1 of the above = *Mult-factor authentication*
Supporting mechanisms

These relate identities ("principals") to actions

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Authorization</th>
<th>Audit-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can a system tell <em>who a user is</em></td>
<td>How can a system tell <em>what a user is allowed to do</em></td>
<td></td>
</tr>
<tr>
<td>What we know</td>
<td>Access control policies (defines)</td>
<td>-</td>
</tr>
<tr>
<td>What we have</td>
<td>+ Mediator (checks)</td>
<td>-</td>
</tr>
<tr>
<td>What we <em>are</em></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>&gt;1 of the above =</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td><em>Mult-factor authentication</em></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
Supporting mechanisms

These relate identities (“principals”) to actions

**Authentication**
How can a system tell *who a user is*

**Authorization**
How can a system tell *what a user is allowed to do*

**Audit-ability**
How can a system tell *what a user did*

What we know
What we have
What we are
>1 of the above = *Mult-factor authentication*

Access control policies (defines)
+ *Mediator* (checks)
# Supporting mechanisms

These relate identities ("principals") to actions

<table>
<thead>
<tr>
<th>Authentication</th>
<th>Authorization</th>
<th>Audit-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can a system tell <em>who a user is</em></td>
<td>How can a system tell <em>what a user is allowed to do</em></td>
<td>How can a system tell <em>what a user did</em></td>
</tr>
</tbody>
</table>

**What we know**
- What we **have**
- What we **are**

**What we have**
- >1 of the above = *Mult-factor authentication*

**Access control policies**
- (defines)
- + *Mediator* (checks)

**Retain enough info to determine the circumstances of a breach**
Defining Security Requirements

• Many processes for deciding security requirements
Defining Security Requirements

• Many processes for deciding security requirements

• Example: **General policy concerns**
Defining Security Requirements

• Many processes for deciding security requirements

• Example: General policy concerns
  • Due to regulations/standards (HIPAA, SOX, etc.)
Defining Security Requirements

• Many processes for deciding security requirements

• Example: **General policy concerns**
  • Due to **regulations**/standards (HIPAA, SOX, etc.)
  • Due **organizational values** (e.g., valuing privacy)
Defining Security Requirements

• Many processes for deciding security requirements

• Example: **General policy concerns**
  • Due to **regulations**/standards (HIPAA, SOX, etc.)
  • Due **organizational values** (e.g., valuing privacy)

• Example: **Policy arising from threat modeling**
Defining Security Requirements

• Many processes for deciding security requirements

• Example: **General policy concerns**
  • Due to **regulations**/standards (HIPAA, SOX, etc.)
  • Due **organizational values** (e.g., valuing privacy)

• Example: **Policy arising from threat modeling**
  • Which **attacks** cause the **greatest concern**?
    - Who are the likely adversaries and what are their goals and methods?
Defining Security Requirements

• Many processes for deciding security requirements

• Example: General policy concerns
  • Due to regulations/standards (HIPAA, SOX, etc.)
  • Due organizational values (e.g., valuing privacy)

• Example: Policy arising from threat modeling
  • Which attacks cause the greatest concern?
    - Who are the likely adversaries and what are their goals and methods?
  • Which attacks have already occurred?
    - Within the organization, or elsewhere on related systems?
Abuse Cases
Abuse Cases

- Abuse cases illustrate security requirements
Abuse Cases

• Abuse cases illustrate security requirements

• Where use cases describe what a system *should* do, abuse cases describe what it should *not* do
Abuse Cases

• Abuse cases illustrate security requirements

• Where use cases describe what a system should do, abuse cases describe what it should not do

• Example use case: The system allows bank managers to modify an account’s interest rate
Abuse Cases

• Abuse cases illustrate security requirements

• Where use cases describe what a system *should* do, abuse cases describe what it *should not* do

• Example use case: The system allows bank managers to modify an account’s interest rate
• Example abuse case: A user is able to spoof being a manager and thereby change the interest rate on an account
Defining Abuse Cases
Defining Abuse Cases

- Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement.
Defining Abuse Cases

• Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement
• Based on the threat model
Defining Abuse Cases

- Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement.
  - Based on the threat model.
  - What might occur if a security measure was removed?
Defining Abuse Cases

- Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement
- Based on the threat model
- What might occur if a security measure was removed?

**Example:** Co-located attacker steals password file and learns all user passwords
Defining Abuse Cases

• Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement
  • Based on the threat model
  • What might occur if a security measure was removed?

• Example: Co-located attacker steals password file and learns all user passwords
  • Possible if password file is not encrypted
Defining Abuse Cases

• Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement
  • Based on the threat model
  • What might occur if a security measure was removed?

  • **Example**: Co-located attacker steals password file and learns all user passwords
    • Possible if password file is not encrypted
  • **Example**: Snooping attacker replays a captured message, effecting a bank withdrawal
Defining Abuse Cases

- Using attack patterns and likely scenarios, construct cases in which an adversary’s exercise of power could violate a security requirement
  - Based on the threat model
  - What might occur if a security measure was removed?

- **Example**: Co-located attacker steals password file and learns all user passwords
  - Possible if password file is not encrypted
- **Example**: Snooping attacker replays a captured message, effecting a bank withdrawal
  - Possible if messages are have no nonce
Security design principles
Design Defects = Flaws

- Recall that software defects consist of both flaws and bugs
  - **Flaws** are problems in the design
  - **Bugs** are problems in the implementation

- **We avoid flaws during the design phase**

- According to Gary McGraw, **50% of security problems are flaws**
  - So this phase is very important
Categories of Principles
Categories of Principles

• Prevention
Categories of Principles

• Prevention
  • **Goal**: Eliminate software defects entirely
Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java
Categories of Principles

• **Prevention**
  • **Goal**: Eliminate software defects entirely
  • **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

• **Mitigation**
Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects
Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects
  - **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another
Categories of Principles

• **Prevention**
  • **Goal**: Eliminate software defects entirely
  • **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

• **Mitigation**
  • **Goal**: Reduce the harm from exploitation of unknown defects
  • **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another

• **Detection** (and **Recovery**)
Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects
  - **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another

- **Detection (and Recovery)**
  - **Goal**: Identify and understand an attack (and undo damage)
Categories of Principles

• **Prevention**
  • **Goal**: Eliminate software defects entirely
  • **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java

• **Mitigation**
  • **Goal**: Reduce the harm from exploitation of unknown defects
  • **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another

• **Detection (and Recovery)**
  • **Goal**: Identify and understand an attack (and undo damage)
  • **Example**: Monitoring (e.g., expected invariants), snapshotting
Principles for building secure systems

- Security is economics
- Principle of least privilege
- Use fail-safe defaults
- Use separation of responsibility
- Defend in depth
- Take human factors into account
- Ensure complete mediation

- Accept that threat models change
- If you can’t prevent, detect
- Kerkhoff’s principle (no security through obscurity)
- Design security from the ground up
- Prefer conservative designs
- Proactively study attacks
Ensure complete mediation
Use separation of responsibility
Defense in depth
Bank:
password: goMets12

credit card:
bowser 8

brokerage:
iniTial23

e-mail:
letmein

Log in to your message center.
Invalid log in or server error. Please try again.
Forgot your password?

Log In Address
example: joi234@jumbowidgetsco.com
Password
note: password is case-sensitive
Remember my Address and Password (what is this?)

[LOG IN]
Account for human factors (“psychological acceptability”)  
(a) Users must buy into the security  
(b) The system must be usable
Account for human factors
Account for human factors

When you see a dialog box like this, click 'Yes' to make it go away. If available, click the checkbox first to avoid being bothered by it again.

- **In the future, do not show this message.**

[Yes] [No]
Account for human factors
Account for human factors
Kerkhoff’s principle