Applications Security

Secure software development process
Designing for security

- Securing software requires outlining security requirements and integrating security planning into the software development process, from design to implementation and testing.

- In order to capture the security requirements, it is important to do risk analysis.
Caveat

- Both risk analysis and security testing are more art than science. They require deep practical understanding of the software development process as well as of real-world security issues.

- In the next few classes, you will be introduced to some of the general security vulnerability classes that often plague software developed without a good security process.
Some rules-of-thumb

1. **Secure the weakest link first.** Use information from risk analysis. E.g.:
   1. Password management more often a source of vulnerability than use of weak crypto.

2. **Practice defense-in-depth.** Use redundancy. If a measure fails, there should be a fallback. E.g.:
   1. A system keeping highly confidential data could require two forms of authentication, one of which not based on passwords.
   2. Use both firewalls and encryption of data flows in your network.
3. **Fail securely.** Anticipate things that may fail in a secure operation. Less often mentioned, but equally important is **safe initialization**. E.g.:

1. When protocols are upgraded, often older versions are preserved for backward compatibility. An attacker can force the protocol negotiation to fail to force an insecure version to run. Better solution: Design modular clients that automatically retrieve patches/upgrades when they detect a server version change.

2. If a secure server crashes, does it generate a “core dump” to lesser-privileged system files. If so, cryptographic keys may be compromised.
Rules-of-thumb (3)

4. **Follow the principle of least privilege.** E.g.:
   1. Services should run with only the exact privileges to achieve their goals. For instance, a web server should have the ability to read web files and write to specifically designated directories. It should never run with administrative privileges.
   2. When writing a program, if you need to open an object with read privileges, do not use read-write.

5. **Compartmentalize.** Modularity in combination with segregation of responsibilities and privileges facilitates access control to objects.
Rules-of-thumb (4)

6. **Keep it simple.** *Compartmentalization* and *least-privilege* tend to make code more difficult to design and implement. Strike a balance with a view that simpler code has fewer bugs.
   1. Also, design with a view for *usability*, since users can often ruin a security scheme that requires more than trivial cooperation from them.

7. **Promote privacy.** A common security fallacy is that by asking a lot of identifying information, you get better authentication.
   1. E.g.: If high-value identifiers (e.g., social-security numbers) are used for low-value transactions (e.g., signing in at the front desk) there is a great risk of identity theft and harm. The same is true in computer systems. Substitute low-level identifiers (but be careful w/ usability) whenever possible.
Rules-of-thumb (5)

8. **Hiding secrets** in software is hard.
   
   1. Many software applications embed secrets (e.g., a registration key) in the software. Obfuscating secrets (even in binary code) is difficult, and such secrets often leak. Software also sometimes try to hide functionality (behavior) by obfuscating. Unless stronger techniques than currently in practice become available, obfuscation provides little protection.
   
   2. Consider using incentives + punishment. For instance, some products must use a unique key when downloading patches. The system does not allow too many upgrades on the same key. Most corporations and users of such systems prefer to buy licenses, even if they could find registration keys online, because of the extra management effort with the pirated keys.
9. **Be reluctant to trust.** Do not make unwarranted assumptions about the environment.
   1. E.g.: If you are manipulating keys, once they are no longer necessary, erase them from memory by explicitly assigning the value 0 to the variables. Operating systems usually do not erase data on de-allocation, and will assign the memory cells to another process with the sensitive data still there.

10. **Use your community resources.**
    1. Favor tried-and-true methods, and/or publicly reviewed, standardized protocols. Consult CERT and other public repositories of security knowledge. Network with professionals to keep up with the state-of-knowledge.
Software life-cycle for survivability

CERT’s modifications to software life cycle models for capturing survivability requirements.
Each step must have a survivability component

<table>
<thead>
<tr>
<th>Life-cycle activity</th>
<th>Survivab. element</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>mission definition</td>
<td>Analysis of mission criticality, failure effects</td>
<td>Cost impact estimation of DoS attacks</td>
</tr>
<tr>
<td>...</td>
<td>Survivability requirements from mission perspective</td>
<td>Availability requirements for critical system assets during attacks</td>
</tr>
<tr>
<td>requirements definition</td>
<td>application of survivability coding techniques</td>
<td>Definition of methods to avoid buffer overflow vulnerabilities</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>system implementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Security Testing & Assessment
Security testing

- Empirically testing a live system or system prototype to discover potential vulnerabilities.
  - Example: If the program crashes when given a very long input, there may be opportunities for a buffer overflow exploit.

- Security testing is more effective when there is deep understanding of the security implications of the architecture of the system, something that should have been acquired during the risk analysis phase.
Black box testing

- The system is tested under abnormal conditions (such as long inputs). If unexpected behavior ensues, further inspection is used to identify possible sources of vulnerabilities.

- Not a very effective approach. Many serious security vulnerabilities involve subtle errors that are only evident from very deliberate expert white-box (i.e., code inspection) analysis.

- Often used by IT development teams.
Red teams

- Involves consulting an external group of security experts
  - Red team is given partial knowledge of the software specifications (those an attacker is likely to obtain).
  - The red team tries to attack the system (penetration testing).
- More effective than black box testing
  - Less effective and cost-efficient than providing the same security experts with full knowledge of your code.
  - Due to economic limits on the time the red team can dedicated to this task, it is likely to uncover only a fraction of exploitable vulnerabilities.
Formal security assessment

- The Common Criteria
  - Formally specifies standards for code development, review, and testing, in order to satisfy a particular security profile.
  - The US profile descends from the US DoD Trusted Computer System Evaluation Criteria (also known as the Orange Book.)
  - The Orange book was not really developed for networked machines, so it is partially obsolete.
Formal assessment limitations

- Standards-based approaches list things to accomplish, without specifying how.
  - Software may undergo a standard-conforming process and still not be sufficiently secure for some applications.

- Complacency is the enemy of security. Good processes should preferably complement, rather than substitute for, available talent.
  - Organizations that develop highly critical applications require security professionals that are not only skilled, but also creative.