Principles for Secure Software

- Following these doesn't guarantee security
- But they touch on the most commonly seen security problems
- Thinking about them is likely to lead to more secure code

1. Secure the Weakest Link

- Don’t consider only a single possible attack
- Look at all possible attacks you can think of
- Concentrate most attention on most vulnerable elements

For Example,

- Those attacking your web site are not likely to break transmission cryptography
  - Switching from DES to AES probably doesn't address your weakest link
- Attackers are more likely to use a buffer overflow to break in
  - And read data before it's encrypted
  - Prioritize preventing that

2. Practice Defense in Depth

- Try to avoid designing software so failure anywhere compromises everything
- Also try to protect data and applications from failures elsewhere in the system
- Don’t let one security breach give away everything

For Example,

- You write a routine that validates all input properly
- All other routines that are supposed to get input should use that routine
- Worthwhile to have those routines also do some validation
  - What if there's a bug in your general routine?
  - What if someone changes your code so it doesn't use that routine for input?

3. Fail Securely

- Security problems frequently arise when programs fail
- They often fail into modes that aren't secure
- So attackers cause them to fail
  - To see if that helps them
- So make sure that when ordinary measures fail, the fallback is secure

For Example,

- A major security flaw in typical Java RMI implementations
- If server wants to use security protocol client doesn't have, what happens?
  - Client downloads it from the server
  - Which it doesn't trust yet . . .
- Malicious entity can force installation of compromised protocol

4. Use Principle of Least Privilege

- Give minimum access necessary
- For the minimum amount of time required
- Always possible that the privileges you give will be abused
  - Either directly or through finding a security flaw
- The less you give, the lower the risk

For Example,

- Say your web server interacts with a backend database
- It only needs to get certain information from the database
  - And uses access control to determine which remote users can get it
- Set access permissions for database so server can <u>only</u> get <u>that</u> data
- If web server hacked, only part of database is at risk

5. Compartmentalize

- Divide programs into pieces
- Ensure that compromise of one piece does not automatically compromise others
- Set up limited interfaces between pieces
  - Allowing only necessary interactions

For Example,

- Traditional Unix has terrible compartmentalization
  - Obtaining root privileges gives away the entire system
- Redesigns that allow root programs to run under other identities help
  - E.g., mail server and print server users
- Not just a problem for root programs
  - E.g., web servers that work for many clients
- Research systems like Asbestos allow finer granularity compartmentalization

6. Value Simplicity

- Complexity is the enemy of security
- Complex systems give more opportunities to screw up
- Also, harder to understand all “proper” behaviors of complex systems
- So favor simple designs over complex ones

For Example,

- Re-use components when you think they're secure
- Use one implementation of encryption, not several
  - Especially if you use "tried and true" implementation
- Build code that only does what you need
  - Implementation of exactly what you need are safer than "Swiss army knife" approaches
- Choose simple algorithms over complex algorithms
  - Unless complex one offers necessary advantages
  - "It's somewhat faster" usually isn't a necessary advantage
  - And "it's a neat new approach" definitely isn't

Especially Important When Human Users Involved

- Users will not read documentation
  - So don't rely on designs that require that
- Users are lazy
  - They'll ignore pop-ups and warnings
  - They will prioritize getting the job done over security
  - So designs requiring complex user decisions usually fail

7. Promote Privacy

- Avoid doing things that will compromise user privacy
- Don't ask for data you don't need
- Avoid storing user data permanently
  - Especially unencrypted data
- There are strong legal issues related to this, nowadays

For Example,

- Google's little war driving incident
- They drove around many parts of the world to get information on Wifi hotspots
- But they simultaneously were sniffing and storing packets from those networks
- And gathered a lot of private information
- They got into a good deal of trouble . . .

8. Remember That Hiding Secrets is Hard

- Assume anyone who has your program can learn <u>everything</u> about it
- "Hidden" keys, passwords, certificates in executables are invariably found
- Security based on obfusticated code is always broken
- Just because you're not smart enough to crack it doesn't mean the hacker isn't, either

For Example,

- Passwords often “hidden” in executables
  - GarretCom network switches tried to do this in SCADA control systems
  - Allowed escalation of privilege if one had any login account
- Android apps containing private keys are in use (and are compromised)
- Ubiquitous in digital rights management
  - And it never works

9. Be Reluctant to Trust

- Don't automatically trust things
  - Especially if you don't have to
- Remember, you're not just trusting the honesty of the other party
  - You're also trusting their caution
- Avoid trusting users you don't need to trust, too
  - Doing so makes you more open to social engineering attacks

For Example,

- Why do you trust that shrinkwrapped software?
- Or that open source library?
- Must you?
- Can you design the system so it's secure even if that component fails?
- If so, do it

10. Use Your Community Resources

- Favor widely used and respected security software over untested stuff
  - Especially your own . . .
- Keep up to date on what's going on
  - Not just patching
  - Also things like attack trends

For Example,

- Don’t implement your own AES code
- Rely on one of the widely used versions
- But also don’t be too trusting
  - E.g., just because it’s open source doesn’t mean it’s more secure