Secure Design Principles

Slides adapted from "Foundations of Security: What Every Programmer Needs To Know" by Neil Daswani, Christoph Kern, and Anita Kesavan (ISBN 1590597842; http://www.foundationsofsecurity.com). Except as otherwise noted, the content of this presentation is licensed under the Creative Commons 3.0 License.

Agenda

- Principle of Least Privilege
- Defense-in-Depth & Diversity-in-Defense
- Secure the Weakest Link
- Fail-Safe Stance
- Secure by Default
- Simplicity & Usability

3.1. Principle of Least Privilege

- Just enough authority to get the job done.
- Common world ex: Valet Keys
  - Valets can only start car and drive to parking lot
- Highly elevated privileges unnecessary
  - Ex: valet key shouldn't open glove compartment
  - Web server Ex: can read, not modify, html file
  - Attacker gets more power, system more vulnerable

3.1. SimpleWebServer Example

- If SWS run under root account, clients could access all files on system!
- `serveFile()` method creates `FileReader` object for arbitrary pathname provided by user
  - `GET ../../../../etc/shadow HTTP/1.0`
  - Traverses up to root, `/etc/shadow` on UNIX contains list of usernames & encrypted passwords!
  - Attacker can use this to launch a dictionary attack
  - Need to canonicalize and validate pathname
- Obey Least Privilege: Don't run server under root!

3.1. Canonicalizing Pathnames

- `checkPath()` method: ensure target path is below current path and no .. in pathname

```
String checkPath (String pathname) throws Exception {
    File target = new File(pathname);
    File cwd = new File(System.getProperty("user.dir"));
    /* User's current working directory stored in cwd */
    String targetStr = target.getCanonicalPath();
    String cwdStr = cwd.getCanonicalPath();
    if (!targetStr.startsWith(cwdStr))
        throw new Exception("File Not Found");
    else return targetStr;
}
```

- Then `serveFile()` uses normalized path:

```
fr = new FileReader (checkPath(pathname));
```

3.2. Defense-in-Depth

- Also called redundancy/diversity: layers of defense, don’t rely on any one for security

- Examples
  - Banks: Security Guards, Bullet-Proof, Teller Window, Dye on Money
  - Many different types of magic and many levels of defense protecting the Sorcerer's Stone in Harry Potter

3.2.1. Prevent, Detect, Contain, and Recover

- Should have mechanisms for preventing attacks, detecting breaches, containing attacks in progress, and recovering from them
- Detection particularly important for network security since it may not be clear when an attack is occurring

3.2.2. Don't Forget Containment and Recovery

- Preventive techniques not perfect; treat malicious traffic as a fact, not exceptional condition
- Should have containment procedures planned out in advance to mitigate damage of an attack that escapes preventive measures
  - Design, practice, and test containment plan
  - Ex: If a thief removes a painting at a museum, the gallery is locked down to trap him.

3.2.3. Password Security Example

- Sys Admins can require users to choose strong passwords to prevent guessing attacks
- To detect, can monitor server logs for large # of failed logins coming from an IP address and mark it as suspicious
- Contain by denying logins from suspicious IPs or require additional checks (e.g. cookies)
- To recover, monitor accounts that may have been hacked, deny suspicious transactions

3.3. Diversity-in-Defense

- Using multiple heterogeneous systems that do the same thing
  - Use variety of OSes to defend against virus attacks
  - Second firewall (different vendor) between server & DB
- Cost: IT staff need to be experts in and apply to patches for many technologies
  - Weigh extra security against extra overhead

3.4. Securing the Weakest Link

- "Information System is only as strong as its weakest link."
- Common Weak Links:
  - Unsecured Dial-In Hosts: War Dialers (historical)
  - Weak Passwords: easy to crack
  - People: Social Engineering Attacks
  - Buffer Overflows from garbage input

3.4.1. Weak Passwords

- One-third of users choose a password that could be found in the dictionary
- Attacker can employ a dictionary attack and will eventually succeed in guessing someone's passsword
- By using Least Privilege, can at least mitigate damage from compromised accounts

3.4.2. People

- Employees could fall for phishing attacks (e.g. someone calls them pretending to be the "sys admin" and asks for their password)
  - Especially a problem for larger companies
- Malicious Programmers
  - Can put back doors into their programs
  - Should employ code review
- Keep employees happy, less incentive for them to defraud company
  - Also distribute info on need-to-know basis, perform background checks on hires

3.4.3. Implementation Vulnerabilities

- Correct Design can have bugs in implementation
- Misuse of encryption can allow attacker to bypass it and access protected data
- Inadvertent mixing of control and data
  - Attacker feeds input data that's interpreted as a command to hijack control of program
  - Ex: buffer overflows, SQL injection

3.5. Fail-Safe Stance

- Expect & Plan for System Failure
- Common world example: Elevators
  - Designed with expectation of power failure
  - In power outage, can grab onto cables or guide rails
- Ex: If firewall fails, let no traffic in
  - Deny access by default
  - Don't accept all (including malicious), because that gives attacker additional incentive to cause failure

3.5.1. SWS Fail-Safe Example

```
public void serveFile (OutputStreamWriter osw,
          String pathname) throws Exception {
   FileReader fr=null;
   int c=-1;
   StringBuffer sb = new StringBuffer();
   /* ...code excluded... */
   while (c != -1) {
       sb.append((char)c); // if memory run out, crashes!
       c = fr.read();
   }
   osw.write (sb.toString());
```

- Crashes, but doesn't do something insecure
- Still a bug since it can be used for DoS
  - Attacker could use `/dev/random`, infinite length file

3.5.2. Checking the File Length

- One fix: have a default maximum amount of data to read from file
  - Only serve file if sufficient memory available

```
pathname = checkPath(pathname); // canonicalize
File f = new File (pathname);
/* ... */
if (f.length() > Runtime.getRuntime().freeMemory()) {
             throw new Exception();
}
```

- Still doesn't work for `/dev/random`, since it's a special file whose length is reported as 0 (it doesn't actually exist on disk)

3.5.3. Don't Store the File in Memory

- Instead of storing the bytes of the file before sending it, just stream it

```
while (c != -1) {
     osw.write(c); // No StringBuffer storage
     c = fr.read();
}
```

- Problem: `/dev/random` causes server to be forever tied up servicing attacker's request, can't serve other legitimate requests (DoS still possible)

3.5.4. …and Impose a Download Limit

- To properly defend against `/dev/random` attack, need to impose max download limit

```
while ((c != -1) && (sentBytes < MAX_DOWNLOAD_LIMIT)) {
      osw.write (c);
      sentBytes++;
      c = fr.read();
}
```

- Tradeoff: limit too low, legitimate files get truncated; limit too high, DoS still a threat from abusive requests

3.6. Secure By Default

- Only enable 20% of products features that are used by 80% of user population
- “Hardening” a system: All unnecessary services off by default
- More enabled features means more potential exploits and decreased security
- Example: Windows OS
  - all features turned on to make users hooked
  - there were lot of viruses like Code Red and Nimda which exploited IIS vulnerability

3.7. Simplicity

- Security holes likely in complex software
- Simpler design is easier to understand and audit
- *Choke point*: centralized piece of code through which all control must pass
  - keeps security checks localized, easier to test
- Less functionality = Less security exposure

3.8. Usability

- Usable = users can easily accomplish the tasks they need to do with the software
- Don't rely on documentation: enable security features by default, design to be easy to use
  - Difficulty is in tradeoff with user convenience
- Users are lazy (They ignore security dialogs)
  - Prevent users from committing insecure actions, assist them in doing it securely
  - "Why Johnny Can't Encrypt" – "usability for security"

3.8. Usability for Security

- Definition: (Whitten-Tygar) Security software is usable if the people who are expected to use it:
  - are reliably made aware of security tasks they need to perform
  - are able to figure out how to successfully perform those tasks
  - do not make dangerous errors
  - are sufficiently comfortable with the interface to continue using it

3.9. Security Features Do Not Imply Security

- Using one or more security algorithms/protocols will not solve all your problems!
  - Using encryption doesn't protect against weak passwords.
  - Using SSL doesn't protect against buffer overflows.

- Schneier: "Security is a process, not a product!"
  - Can never be completely secure, just provide a *risk assessment* (more testing lessening risk)
  - Attacker only needs to find one flaw, designers have to try and cover all possible flaws
  - Security features can help, but can't stop bugs

Summary

- Employ a few key design principles to make system more secure.
  - Avoid elevated privileges
  - Use layered defense (prevention, detection, containment, and recovery)
  - Secure weakest links
  - Have fail-safes, i.e. crash gracefully
  - Don't enable unnecessary features
  - Keep design simple, usable
  - Security features can't compensate for bugs