CIS 418/518 – Secure Software Engineering
Security Principles

Jagadeesh Nandigam

School of Computing
Grand Valley State University
nandigaj@gvsu.edu

Outline

SD^3 – Secure by Design, by Default, and in Deployment

- Application security must be designed and built into your solutions from the start.
- A set of high-level concepts and strategies that when adopted can help shape the development process to deliver secure systems.
  - Secure by Design
  - Secure by Default
  - Secure in Deployment

Secure by Design

- Take appropriate steps to make sure the overall design of the product is sound from the outset.
  - Assign a "go-to person" for your security issues. This is the person who signs off on the product being secure.
  - Require security training for all personnel.
  - Make sure threat models are in place by the time the design phase is complete.
  - Adhere to design, coding, and testing guidelines.
  - Fix all bugs that deviate from the guidelines as soon as possible.
  - Make sure the guidelines evolve as you learn new vulnerabilities and learn new best practices for mitigating them.
  - Develop regression tests for all previously fixed vulnerabilities.
  - Simplify the code, and simplify your security model.
  - Perform penetration analysis before you ship.

Secure by Default

- The goal of secure by default is to ship a product that is secure enough out of the box.
  - Do not install all features and capabilities by default.
  - Allow least privilege in your application. Don't require your code be used by members of the local or domain administrators group when it does not require such elevated capabilities.
  - Apply appropriate protection for resources.

Secure in Deployment

- Secure in deployment means the system is maintainable (i.e., easy to deploy and administer) once your users install the product.
  - Make sure the application offers a way to administer its security functionality.
  - Create good quality security patches as soon as feasible.
  - Provide information (online help, documentation, or cues on-screen) to the user on how to use the system in a secure manner.

Security Principles

- Security principles for building secure systems:
  - Minimize attack surface area
  - Use defense in depth
  - Use least privilege
  - Establish secure defaults
  - Don't trust external services
  - Fail to a secure mode
  - Avoid security by obscurity
  - Don't mix code and data
  - Fix security issues correctly
  - Separation of duties
  - Keep security simple

Security Principle: Minimize Attack Surface Area

- Keep points of entry to the application to a minimum and allow your users to enable functionality as they need it.
  - Number of open sockets and open named pipes
  - Number of open remote procedure call (RPC) endpoints
  - Number of services
  - Number of services running by default
  - Number of services running in elevated privileges
  - Number of dynamic-content Web pages
  - Number of accounts you add to an administrators group
  - Number of files, directories, and registry keys with weak access control lists (ACLs)

Security Principle: Use Defense in Depth

- Have multiple layers of security controls (defense) to defend a system against any attacks.
- Always be prepared to defend your application from attacks because the security features defending it might be compromised.
- Defense in depth helps reduce the likelihood of a single point of failure in the system.

Security Principle: Use Least Privilege

- All applications should execute with the least privilege to get the job done and no more.
- If a process running as an administrator is compromised (for example, an attacker can inject code into the process, make the code perform sensitive tasks, or run a Trojan horse or virus), the malicious code runs with elevated privilege.
- If your application fails to run unless the user (or service process identity) is an administrator or the system account, determine why. Chances are good that elevated privileges are unnecessary.

Security Principle: Establish Secure Defaults

- The less often used features should be off by default to reduce potential security exposure. If a feature is not running, it cannot be vulnerable to attack.
- 80–20 rule: 20% of the product is used by 80 % of the users.
- The 20% feature set is on by default, and the 80% feature set is off by default with simple instructions and menu options for the enabling of features.
- Another reason for not enabling all features by default has to do with performance. Enabling all features by default not only increases its attack surface but also leads to performance degradation.

Security Principle: Don't Trust External Services

- Assume any data you receive from an external system you do not have complete control over to be insecure and a source of attack.
- External servers can also be a potential point of attack.
- Identify a list of trusted and untrusted data sources for your system. Be very wary of data that flows into a trusted process from an untrusted source.

Security Principle: Fail to a Secure Mode

- Failing securely means the application has not disclosed any data that would not be disclosed ordinarily, that the data still cannot be tampered with, and so on.
- When you fail, do not issue huge swaths of information explaining why the error occurred. Give just enough information so that the user knows the request failed, and log the details to a secure log file.
- The golden rule when failing securely is to deny by default and allow only once you have verified the conditions to allow.

Security Principle: Fail to a Secure Mode

```
isAdmin = true;
try {
   codeWhichMayFail();
   isAdmin = isUserInRole("Administrator");
} catch (Exception ex) {
   log.write(ex.toString());
}
```

If either `codeWhichMayFail()` or `isUserInRole()` fails or throws an exception, the user is an admin by default – an obivous security risk.

Security Principle: Avoid Security by Obscurity

- Always assume that an attacker knows everything that you know, even if this is not true.
- Obscurity is a useful defense, so long as it is not your only defense.

Security Principle: Don't Mix Code and Data

- Mixing code and data is a serious security issue.
- Once you add code to the data, that data becomes dangerous.
  - Number of virus issues that come through e-mail because the e-mail message mixes data (the e-mail message) with code (in the form of script and attachments).
  - Web page security issues, such as cross-site scripting flaws, exist because HTML data and Javascript code are commingled.

Security Principle: Fix Security Issues Correctly

- If you find a security code bug or a design flaw, fix it and go looking for similar issues in other parts of the application.
- If you encounter a common flaw pattern, take steps to add defensive mechanisms that reduce the class of issues.
- If you find a security bug, make the fix as close as possible to the location of the vulnerability.
- If a security flaw exists, fix the root of the problem, not just its symptoms.

Security Principle: Separation of Duties

- A key fraud control is separation of duties.
- Certain roles have different levels of trust than normal users – administrators vs normal users of an application
- In general, administrators should not be users of the application. An administrator should log in as a normal user to do tasks that normal users carry out using the application.

Security Principle: Keep Security Simple

- Attack surface area and simplicity (of the system and the security model used) go hand in hand.
- Have plans in place to simplify old code by shedding unused and insecure featuers over time.
- Avoid code degeneration.

References

- Michael Howard and David LeBlanc, *Writing Secure Code*, Chapter 3 on *Security Principles to Live By*, Microsoft.
- Security by Design Principles at `https://www.owasp.org/index.php/Security_by_Design_Principles`