Secure Coding Practices

COMP620

CERT Secure Coding Initiative

- Works with software developers and software development organizations to reduce vulnerabilities resulting from coding errors
- Many of the slides in this presentation come from the CERT list of the Top 10 Secure Coding Practices

Define security requirements

- Identify and document security requirements early in the development life cycle
- Make sure that code is evaluated for compliance with those requirements
- When security requirements are not defined, the security of the resulting system cannot be effectively evaluated

Model Threats

- Use threat modeling to anticipate the threats to which the software will be subjected
- Identify and categorize the threats to each asset or component
- Rate the threats based on a risk ranking
- Develop a threat mitigation strategy that is implemented in design, code, and test cases

Validate Input

- Validate input from all untrusted data sources
- Proper input validation can eliminate the vast majority of software vulnerabilities
- Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files
- Simple checks of method parameters can help detect errors earlier and easier

Sanitize Data Sent to Other Systems

- Check the validity of any data sent to subsystems such as command shells, relational databases, and third party components
- Attackers may be able to invoke unused functionality in these components
- Because the calling process understands the context, it is responsible for sanitizing the data

Heed Compiler Warnings

- Compile code using the highest warning level available for your compiler
- Eliminate warnings by modifying the code
- Replace potentially dangerous functions with their safer version

Keep It Simple

- Keep the design as simple and small as possible
- Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use
- The effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex

Default Deny

- Base access decisions on permission rather than exclusion
- By default, access should be denied and the protection scheme identifies conditions under which access is permitted

Fail Securely

- If the program encounters an error, it should leave the system in a secure state

```
isAdmin = true;
try {
  codeWhichMayFail();
  if(regular user) isAdmin = false;
}
catch (Exception ex) {
    log.write(ex.toString());
}
```

Principle of Least Privilege

- Every process should execute with the least set of privileges necessary to complete the job
- Any elevated permission should be held for a minimum time
- This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges

External Systems are Insecure

- Do not trust data from users
  - Test input data
  - Avoid parameter tampering
- Third party systems may be compromised

Practice Defense in Depth

- Create systems with multiple defensive strategies
- If one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability or limit the consequences of a successful exploit

Fix Security Issues Correctly

- All too often the quick fix for an immediate problem becomes the final fix
- Problems corrected with valid solutions that follow the architecture of the system are more likely to provide strong security

Use Quality Assurance Techniques

- Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities
- Penetration testing, fuzz testing, and source code audits should be incorporated as part of an effective quality assurance program
- Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective

Fuzz Testing

- Software testing technique that provides invalid, unexpected, or random data to the inputs of a program
- Most valuable when testing input format from an external source, such as files downloaded from the web
- Generally only finds very simple faults

Code Checking Tools

- There are many tools designed to check your program for well known flaws
- Static code checkers read the source code of your program and report on potential vulnerabilities
- Dynamic code checkers execute the program with a variety of data

Popular Code Checkers

- Rats – Rough Auditing Tool for Security
  - open source static code checker
  - acquired by Fortify Software Inc.
- Flawfinder
  - runs under Linux
- Fortify
  - Commercial product
- yasca – Yet Another Source Code Analyzer
  - Uses rats