ENPM 809W Introduction to Secure Software Engineering

Gananand Kini

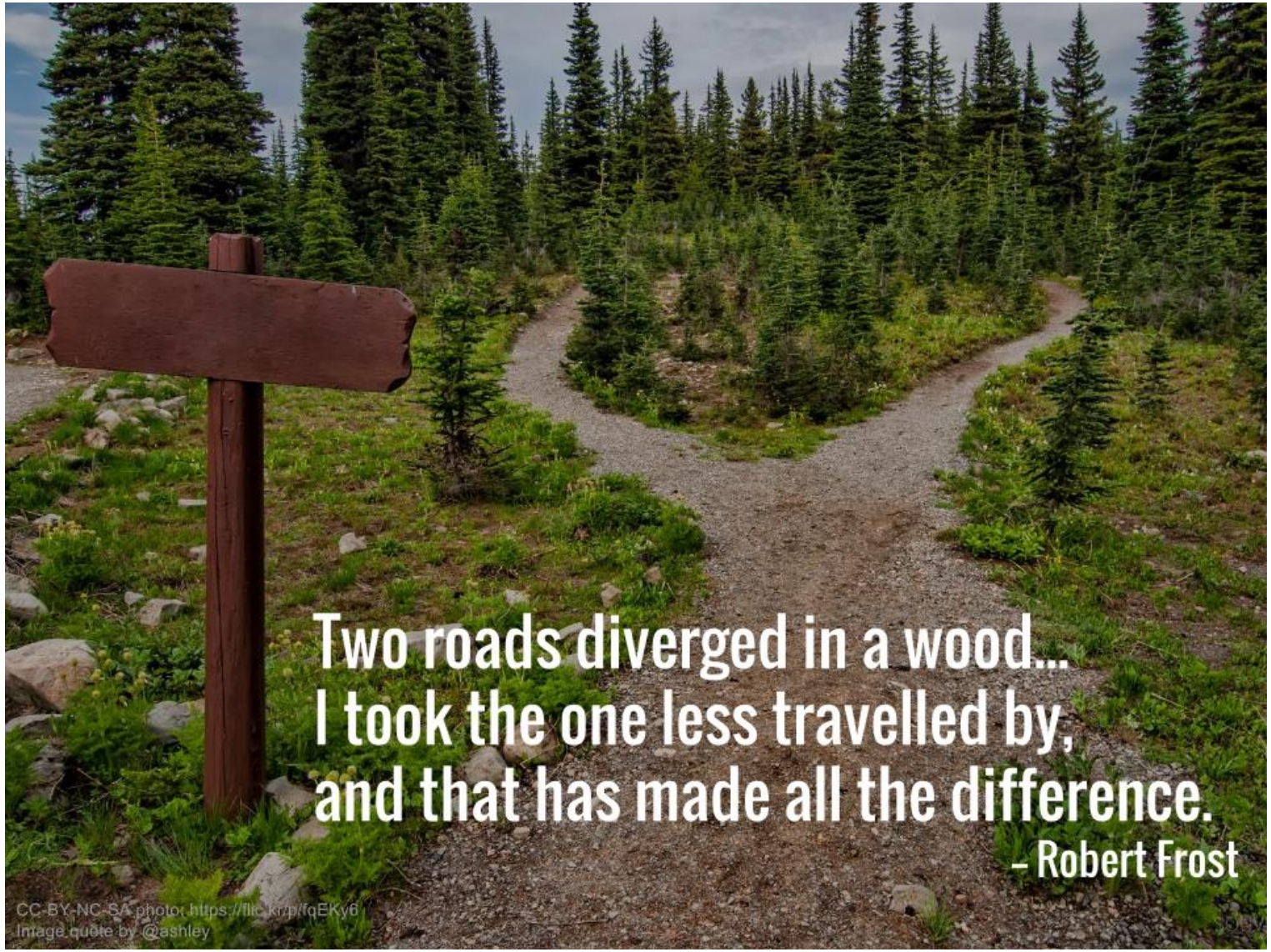
Review and Questions

MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

Outline

- **Review of topics covered in the course**

The road less traveled …



Sources:
1. "You Choose Your Path" (CC BY-NC-SA 2.0) by `James Wheeler

Review

Security Principles

- Security objectives: Confidentiality, Integrity, Availability, Non-repudiation, Privacy, Audit/Accountability, Identity
- Defense-in-depth
- Attack Surface Reduction
- Establish Secure Defaults
- Principle of least privilege
- Fail Securely
- Complete Mediation (“Non-bypassable”) security mechanisms

Security Principles contd…

- Don’t trust service interfaces
- Separation of Duties/Privilege
- Avoid Security through Obscurity
- Avoid Non-atomic security operations
- Fix security issues correctly by addressing the root cause
- Principle of least common security mechanism (Common mechanisms depended on by all users)
- Limit resource dependencies
- Software system ease of use (for security features)
- Harden the software system environment as well as the software

Problems to Avoid

- Confused Deputy Problem
- Time-of-check-time-of-Use
- Covert/Side Channels

Threat Modeling

- **Perform threat modeling using STRIDE to avoid security pitfalls in your software system:**
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege

Input Handling

- Validate all untrusted user input.
- Neutralize inputs using validation, filtering, encoding or escaping.
- Don’t form paths completely from user input.
- Use the allow listing approach rather than the deny listing approach to prevent users from specifying resource identifiers.
- Program defensively assuming things are going to fail.

Cryptography

- Secure the necessary data flows.
- Ensure transactional checking of identity and signatures. (Remember the principle of avoiding non-atomic security operations?)
- Only use the direct results from what was checked (e.g. identity, signature).
- Use cryptographically secure random number generators for cryptographic operations.
- Use secure random initialization vectors and manage their lifecycle wisely (don’t reuse same IV).
- Use established libraries for performing cryptographic operations instead of rolling your own.
- Do model cryptographic applications to understand the risks.

Authentication and Authorization

- Store user secrets securely including passwords.
- Use the correct method of storing passwords (choose a strong hash algorithm and implement salt).
- Implement secure password policies.
- Don’t hardcode credentials in the application.
- Prevent users from authenticating repeatedly within a short time frame.
- Add authorization checks to all the key data flows within the software system in order to prevent abuse.
- Don’t just perform client-side checks, instead use both server-side as well as client-side checks for authorization.
- Remember the principle of least privilege when defining authorizations.
- Implement life-cycles for and manage the authorizations (with the ability to revoke if un-necessary or expired).
- Remember to understand how federation works to implement services securely.

Session Management

- Remember that a session is only useful if there is requirement to persist state across multiple operations. If that is not required, don't implement sessions.
- Only store essential data required for the session.
- Enforce lifetimes on sessions.
- Manage the session lifetime by invalidating the session when the principal logs out or ends the series of operations they were authenticated and authorized to perform. This helps prevent session fixation.
- Cross-site request forgery (CSRF/XSRF) is still a hard problem. Prevent stealing of credentials or leaks of authentication/authorization tokens where possible and implement policies that limit the user base for sensitive operations.
- Also remember to apply the defense-in-depth principle here and not just rely on a single mechanism for authentication/authorization.
- Re-authenticate and Re-authorize for sensitive operations.
- Harden headers and restrict all resources accessible as well as any means available by checking for authorization. (Remember the HTTP verb tampering attack?)

Error Handling & Logging

- Do check for errors and handle them if and when they occur.
- Don’t allow users to see the raw error. Rather modify what the user sees so sensitive information is not revealed.
- Don’t reveal sensitive information due to an error condition.
- Clean up resources being held after handling exceptions.
- Handle specific errors rather than general “catch-alls”.
- When possible do log the errors and log defensively. Don’t allow user input to control what goes into the log file.
- Manage the logs and their life-cycles wisely.
- Use existing libraries that can perform log operations for you instead of rolling your own.

Debug Code and Release Proofing

- Remember to remove any logic or code (“back-door code”) that relies on the debug state of the software system.
- Remember to remove sensitive information from debug messages before logging them.
- Always use the platform/language to generate a Release version of the binary instead of a Debug version wherever possible.
- Follow the release process checklist and perform secure code reviews before the final release.

You have …

- **Now understood security principles and how to apply them in software engineering!**
- **Now gained experience in both:**
  - Attacking software systems to understand security consequences as well as
  - Applying those concepts to then defending against those attacks!
- **Learnt and applied how to use:**
  - Git for version control
  - The C# language
  - The ASP.NET framework.
- **Designed and Implemented a software system project**
- **Performed a Secure Code Review**
- **…**

Congratulations on completing the course!