AIT 681
Secure Software Engineering

Topic #6. Seven Software Security Touchpoints (III)
Instructor: Dr. Kun Sun
• This lecture
  — [McGraw]: Ch. 7-9
Seven Touchpoints

1. Code review
2. Architectural risk analysis
3. Penetration testing
4. Risk-based security testing
5. Abuse cases
6. Security requirements
7. Security operations
4. Risk-Based Security Testing
Risk-Based Security Testing

• Artifact: unit and system
• Strategies:
  – Testing of security functionality (standard functional testing)
  – Risk-based security testing (attack pattern, risk analysis, abuse cases)
• Attacker’s mindset
• White Hat + Black Hat
Security Testing beyond Penetration

• Should also include probing software behavior as a critical aspect of system behavior
  – Deeper than simple black box probing
  – Even beyond the functional testing of security requirements.

• Testers must carry out a risk-based approach
  – Grounded in both the system's architectural reality and the attacker's mindset
Pen Testing vs. Security Testing

• Both approaches should take: risk analysis, abuse cases, and functional security requirements

• Major differences:
  • the timing of the testing
  • the level of approach

• Penetration testing:
  – Is performed in an operational environment when the software is completed and installed.
  – Focuses on outside → in.

• Security testing:
  – Can be in unit level at testing environment.
  – Mostly inside → out
• Traditional approaches to computer and network security testing focus on network infrastructure, firewalls, and port scanning.
  – An outside → in perspective
Security Testing

• Security testing should start at the feature or component/unit level
• At design phase, risk is identified and ranked.
• At component level:
  – Risks to the component's assets must be mitigated within the bounds of contextual assumptions.
  – Tests are structured to attempt both unauthorized misuse of and access to target assets
Security Testing (Cont.)

• At system level:
  – Focus on identifying *intra-component* failures and assessing security risk inherent at the design level
  – Data flow diagrams, models, and *inter-component* documentation created during the risk analysis stage can be a great help.
  – Abuse cases developed earlier in the lifecycle should be used to enhance a test plan with adversarial tests based on plausible abuse scenarios.
Risk Management and Security Testing

• A critical aspect of security testing relies on directly probing security risks:
  – Architectural risk analysis
  – Risk-based security test planning
  – Security testing

• Security testing must involve two diverse approaches:
  – Functional security testing: testing security mechanisms to ensure that their functionality is properly implemented.
  – Adversarial security testing: performing risk-based security testing by understanding and simulating the attacker’s approach
How to Approach Security Testing

• Who:
  – Standard testing organizations using a traditional approach can perform functional security testing.
  – Risk-based security testing relies more on expertise and experience

• How:
  – White box analysis involves analyzing and understanding both source code and the design.
  – Black box analysis refers to analyzing a running program by probing it with various inputs.
Malicious Input

• Malicious input is the main way to break a system from the outside
• Input includes:
  – Register settings
  – Environment variables
  – File contents
  – Network configuration.
• Attacker toolkits focus on input: Fault injection tools, grammar generators, and re-players.
Beyond Input

• Testing around malicious input is necessary but not sufficient.
  – Should also focus on data structures, components, APIs, program state, etc.

• Other elements to consider
  – The forest-level view created during architectural risk analysis.
  – Sockets & pipes
  – Files & the Win32 Registry
  – Remote procedure calls (RPCs)
  – Command-line arguments
Time-Based Attack

• Program state and state preservation
  – Some modern software protocols in common use (like HTTP) are stateless.
  – E.g., changing a "hidden" variable in a URL or as complex as de-serializing an object, manipulating it, and re-serializing it.

• Multiple processes:
  – Processes interact and share some kind of data structure.
  – E.g., Time-of-check time-of-use (TOCTOU) race conditions. Multithreaded system
5. Abuse Cases
Abuse Cases

- Artifact: requirements and use cases
- Describe system behavior under attack
- Explicit coverage of
  - What should be protected
  - From whom
  - For how long
- White Hat + Black Hat
Software Development

• Most software design tend to describe features and functions.
  – A completely functional view of a system. E.g., UML, use cases, and other SE model.

• Need to think beyond features, touching on emergent properties of software systems such as reliability, security, and performance.
Holding Software Vendors Accountable

• Legally binding vendors with Service Level Agreements (SLAs).

• SLAs must include expectations of security in the acceptance requirements:
  – Proper implementation of security features
  – Looking for known security flaws and confirming that they are not present
  – Passing third-party validation and verification security tests agreed on in advance
  – Use of source code analysis tools
Security Is Not a Set of Features

• Security is an emergent property of a system, not a feature.
  – Security features and mechanisms, such as cryptography, SSL, alone are not sufficient for building secure software.

• The most cost-effective approach to software security incorporates thinking beyond white hat normative features by wearing a black hat and thinking like a bad guy, and doing this throughout the development process.
Attackers Are Creative

• If the development process doesn't address unexpected or abnormal behavior, an attacker usually has plenty of raw material to work with.
• Designers knows the systems better than the attackers.
• Critical security questions:
  – What assumptions are implicit in our system?
  – What kinds of things would make our assumptions false?
  – What kinds of attack patterns will an attacker bring to bear?
Abuse Cases

• Abuse cases (sometimes called misuse cases or attack pattern): to think as the same way that attackers do
  – By thinking beyond the normative features and functions and also contemplating negative or unexpected events.

• What motivates an attacker?
  – What do I want?
  – How can I accomplish my goal?
Creating Useful Abuse Cases

• A practical method for creating abuse cases is usually through a process of informed brainstorming.
  – This approach relies heavily on experience and expertise.

• Abuse is always possible at the places where legitimate use is possible.
  – User interfaces
  – Functional security requirements
  – Things most developers assume a person can't or won't do.
• One goal of abuse cases is to decide and document a priori how the software should react to illegitimate use.

• The process of specifying abuse cases makes a designer differentiate appropriate use from malicious use.

• Virtually any system that has value can be abused.
Abuse Case Development

• Abuse cases are to be built by a team of requirements people and security analysts.
  – A set of requirements, a set of standard use cases, and a list of attack patterns.

• Microsoft’s STRIDE model list of attack categories:
  – Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.
Creating Anti-Requirements

• A threat is an actor or agent who carries out an attack.
  – Vulnerabilities and risks are not threats.
• Anti-requirements: the things that you don't want your system to do.
• Anti-requirements provide insight into how an attacker (in other words, a threat) can abuse your system.
• It often tied up to the lack of or failure of a security function.
Creating an Attack Model

• An attack model comes by explicit consideration of **known** attacks or attack types.
  – Given a set of requirements and a list of threats.
  – Cycle through a list of known attacks one at a time
    • Think about whether the "same" attack applies to your system

• To create an attack model
  – Select those attack patterns relevant to your system. Build abuse cases around those attack patterns.
  – Include **anyone** who can gain access to the system because threats must **encompass** all potential sources of danger to the system.
What Do We Learn From The Process

• Activities are designed to create
  – A list of threats and their goals.
  – A list of relevant attack patterns
  – A unified attack model

• The process creates a set of ranked abuse cases.

• The more experience and knowledge you have about actual software exploit and real computer security attacks, the more effective you will be at building abuse cases
Abuse Cases Are Useful

• Determining the can'ts and won'ts is often difficult for those who think only about positive features.

• Some guidance exists in the form of attack patterns
  – E.g., the buffer overflow, follows several different standard patterns

• Attack patterns can be used to guide abuse case development.

• Abuse cases is important, but can be overused
6. Security Requirements
Security Requirements

• Artifact: Requirements
• Security explicitly worked into the requirements level
• Both functional security and emergent characteristics
• White Hat
7. Security Operations
Security Operations

- Artifact: fielded system
- Monitoring system usage
- Combines both network centric and software specific operations
- White Hat
External Analysis

• Evaluate security by outside members
• Why?
• Advantages/disadvantages
A Summary of 7 Touchpoints
Best Practices

- Manageable number of simple activities
- Should be applied throughout the software development process
- Problem: a gap between security and development.
  - Software developers: lack of security domain knowledge → limited to functional security
  - Information security professionals: lack of understanding software → limited to reactive security techniques
  - It’s very rare to find information security professionals directly involved in major software development projects.
Abuse Cases

• Drive non-functional requirements and test scenarios
• Need information security professionals to understand attacker’s mind
• Collaboration between software developers and infosec people
Business Risk Analysis

• “Who cares”
• Business risk to stakeholders
• Technology assessment → need software-level assessment
• Answer security related questions: how much down time, cost of recovery, effect on reputation, etc.
Architectural Risk Analysis

- Assess the technical security exposures at system design-level
- Evaluates business impact of technical risks
- Infosec people: understanding of technology, e.g., application platform, frameworks, languages, functions, etc.
- Real world feedback
Security Testing

• In addition to testing functional specifications and requirements, need test for risk-based attacks

• Understand attacker’s way of thinking
Code Review

- Requires knowledge of code
- Need information about attacker’s way of thinking
Penetration testing

• System penetration testing: driven by previously identified risks
  – Outside $\rightarrow$ in activity

• Application penetration testing
  – Inside $\rightarrow$ out activity
Deployment and Operations

• Configuration and customization of software application’s deployment environment

• Activities:
  – Network-component-level
  – Operating system-level
  – Application-level

• Fine tuning security functionality

• Evaluate entire system’s security properties

• Apply additional security capabilities if needed