Imperial College London

CO331 – Network and Web Security

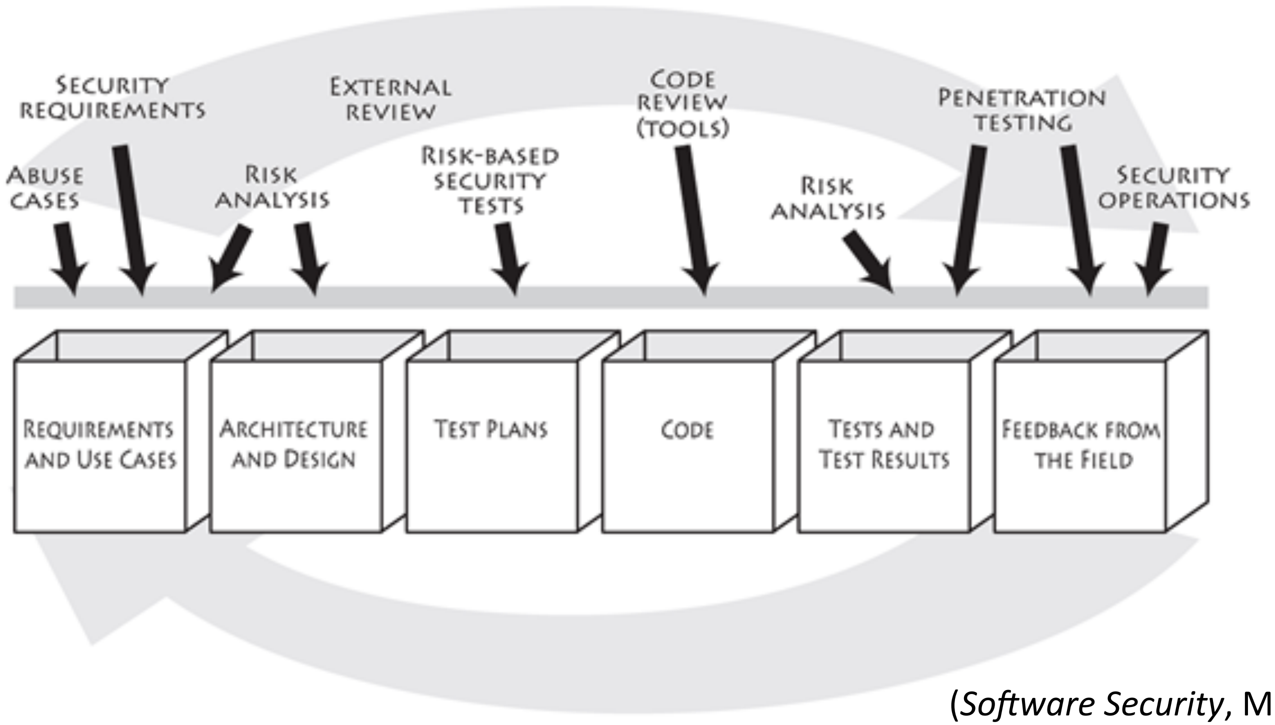
3. SSDLC

Dr Sergio Maffeis
Department of Computing
Course web page: http://www.doc.ic.ac.uk/~maffeis/331

Software engineering

Imperial College London

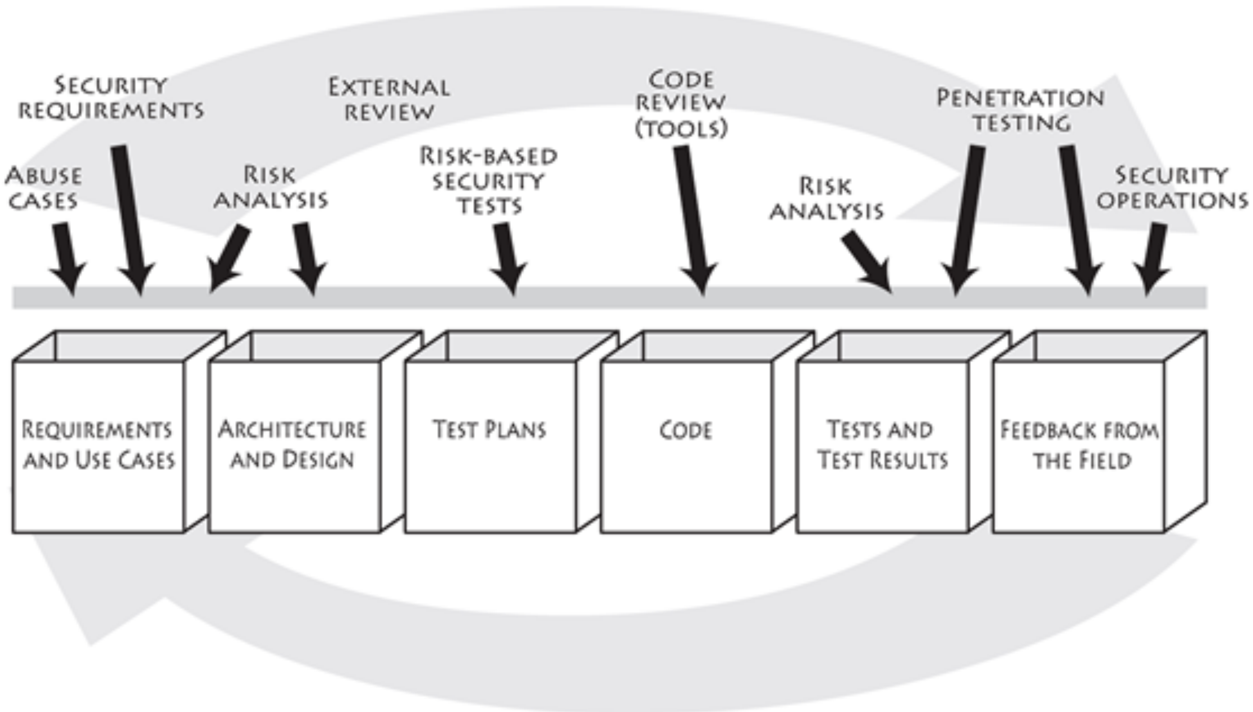
- Security is increasingly getting integrated in software engineering practice
  - Building Security In Maturity Model (BSIMM) http://www.bsimm.com
    - Real-world best practices used by software companies
- *Touchpoints* overlay key **s**ecurity activities on the **s**oftware **d**evelopment **l**ife**c**ycle: SDLC > SSDLC



(*Software Security*, McGraw, 2006)

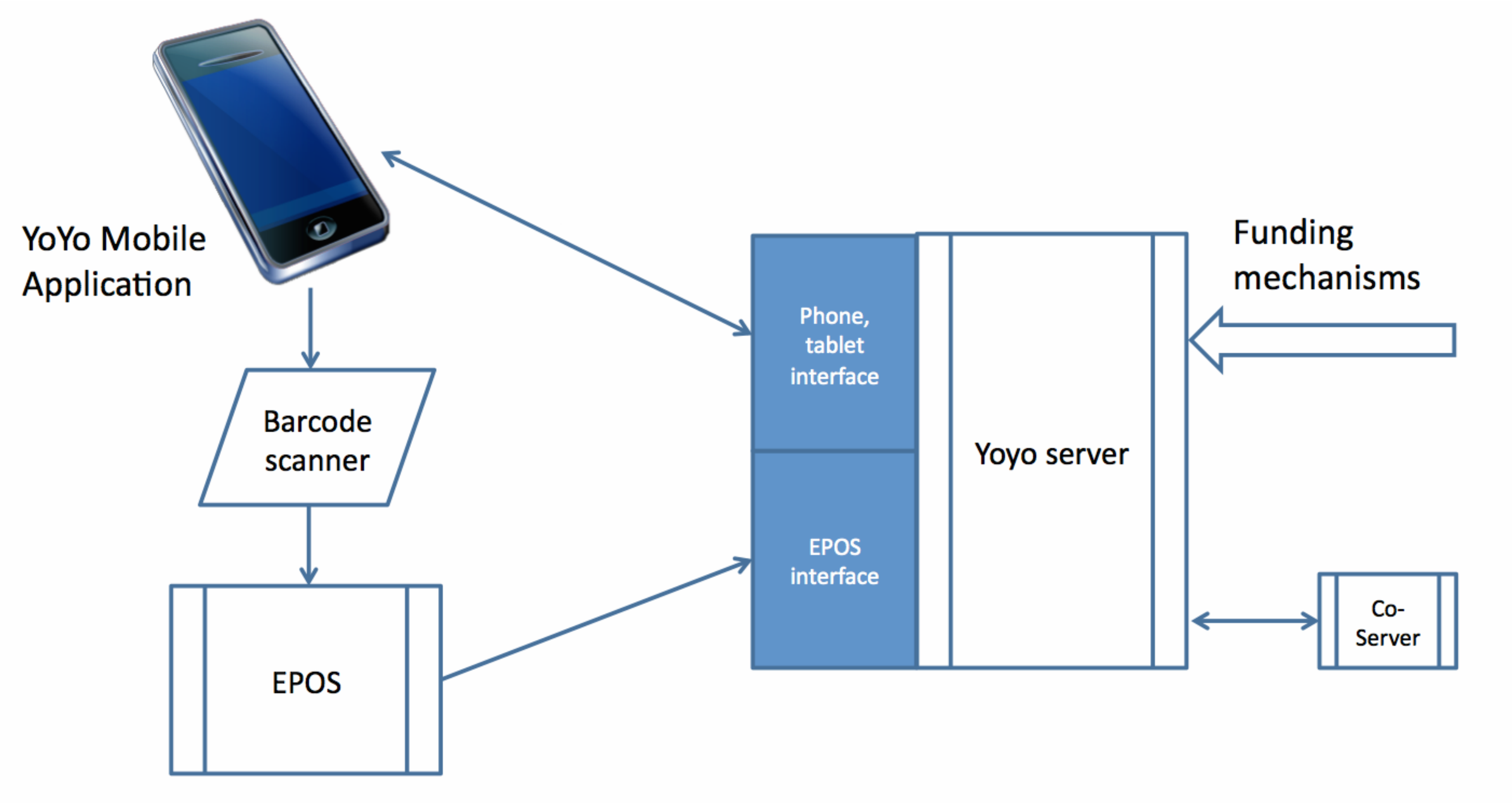
Security touchpoints

- Abuse cases
- Security requirements
- Risk analysis
  - **Threat modelling**
  - Quantitative risk assessment
- Risk-based security tests
- **Code review**
- **Penetration testing**
- Security operations



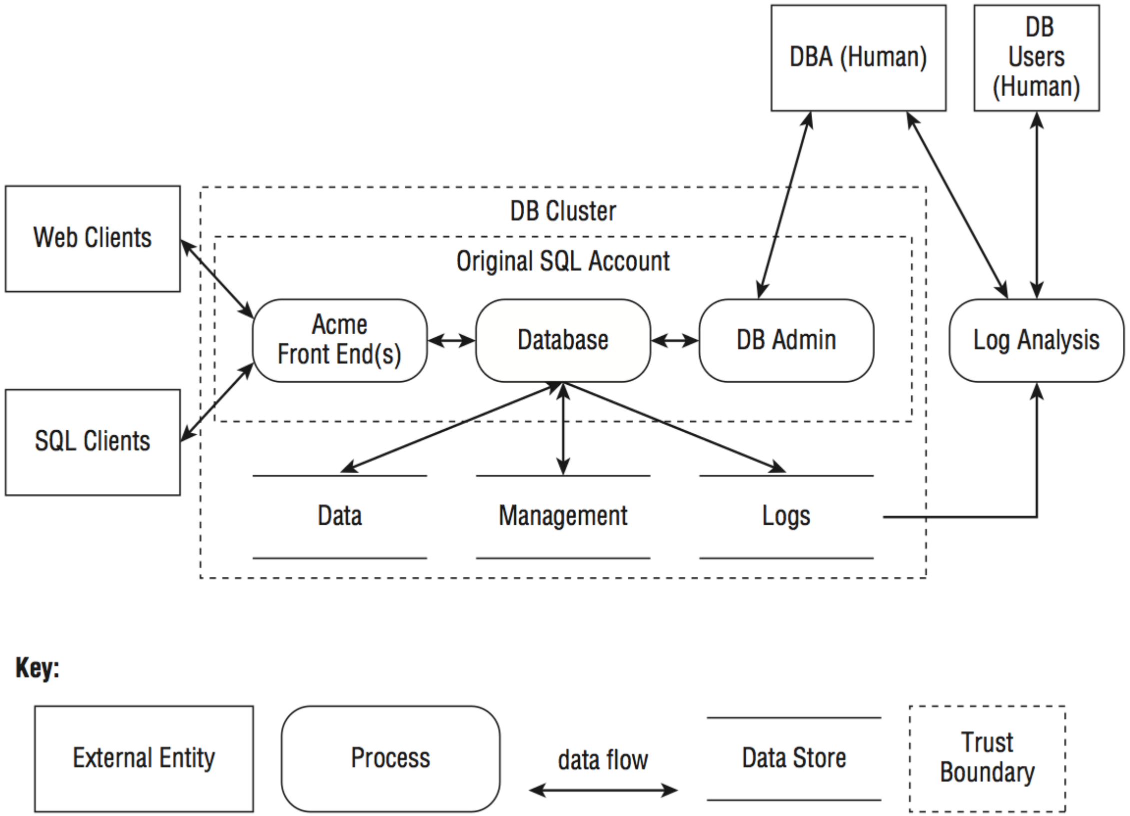
Dive into threat modelling

- Describe some threats to the YoYo payment system



Model the system

- Use consistent visual syntax
- Alternative approaches
  - Focus on assets: password, credit card numbers, …
  - Focus on attackers: hacker, criminal, secret service
  - **Focus on system architecture**
- Data-Flow Diagrams (DFD)
  - Depict flow of information across system components
  - External entities are out of control
  - Trust boundaries help establish what *principal* controls what
  - Attack tend to cross trust boundaries

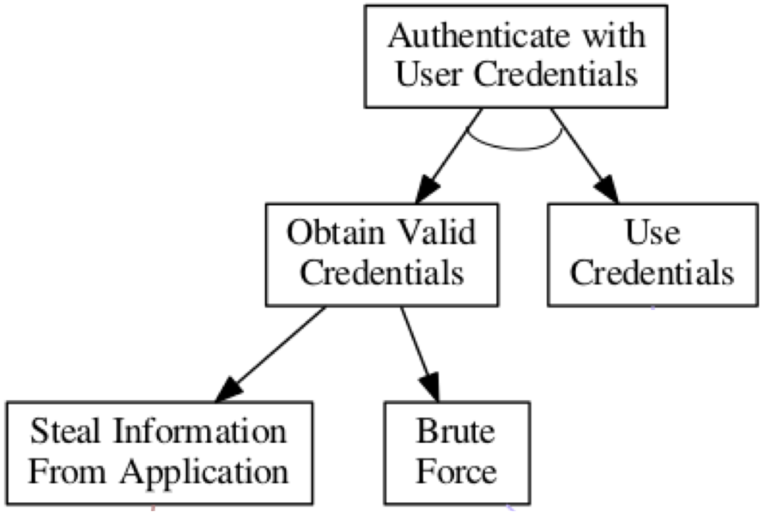


(*Threat Modeling*, Shostack, 2014)

Identify threats: STRIDE

- For each element in a DFD, ask "What can go wrong?"
  - **S**poofing: pretending to be something/somebody else
  - **T**ampering: modifying without permission
  - **R**epudiation: denying to have done something
  - **I**nformation Disclosure: revealing information without permission
  - **D**enial of Service: prevent a system from providing a (timely) service
  - **E**levation of Privilege: achieve to do more than what is intended
- Some threats may belong to more than one category
- Document threats by writing risk-based security tests (where possible).
- *Elevation of Privilege*: a card game based on STRIDE methodology
  - Actually used in practice: https://www.microsoft.com/en-us/SDL/adopt/eop.aspx

Identify threats: attack trees



- Authenticate with credentials
  - Obtain Valid Credentials
    - Steal information from application
    - Brute Force
  - Use credentials

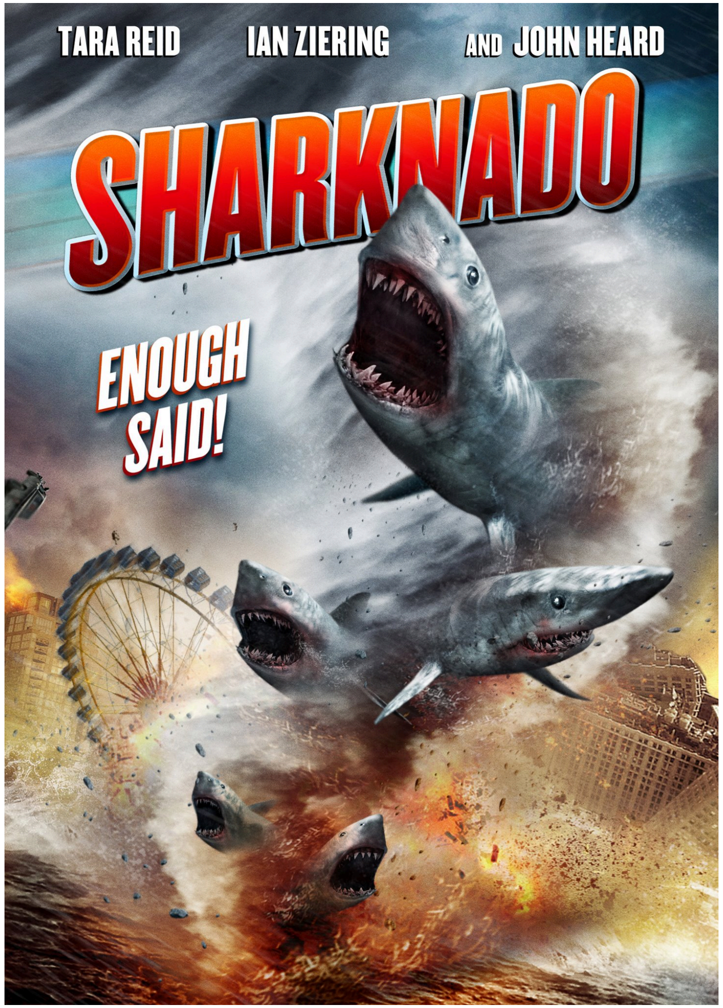
Tree structure

- Root node represent the goal of the attack, or the asset being compromised
- Children are steps to achieve goal
- Leaves are concrete attacks
- By default sibling nodes represent *sufficient* steps to achieve the goal (step 1 **or** step 2)
- Special notation for siblings that represent *necessary* steps (step 1 **and** step 2)
- Trees have alternative textual notation

- Attack trees are an alternative to STRIDE
  - For each element in a DFD, if the goal of an attack tree is relevant, start traversing the tree to identify possible attacks
  - Attack trees capture domain-specific expertise and can be reused on different DFDs

Focus on realistic threats

- Denial of service caused by sharks lifted from ocean by massive tornado
- Nuclear power plant
  - USB stick infected with Stuxnet
  - Earthquake followed by tsunami
- Email account
  - Password-guessing attack
  - Breach in online provider
  - Keylogger on user machine
- What threats should be considered depends on
  - System being modelled
  - Value of the assets being protected
  - Security budget



Evaluate threats

- There are many approaches to evaluating threats
- Beware of formulae that quantify risk
  - It's difficult to estimate realistic parameters
  - Companies don't release breach data, although this is changing
  - Black Swan problem: extremely rare events are hard to predict and quantify
- **DREAD**
  - Score each threat between 5 (lowest) and 15 (highest)
  - Designed at Microsoft, now used in other companies

Evaluate threats

Imperial College London

	Rating	High (3)	Medium (2)	Low (1)
D	Damage potential	The attacker can subvert the security system; get full trust authorization; run as administrator; upload content.	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be reproduced every time and does not require a timing window.	The attack can be reproduced, but only with a timing window and a particular race situation.	The attack is very difficult to reproduce, even with knowledge of the security hole.
E	Exploitability	A novice programmer could make the attack in a short time.	A skilled programmer could make the attack, then repeat the steps.	The attack requires an extremely skilled person and in-depth knowledge every time to exploit.
A	Affected users	All users, default configuration, key customers	Some users, non-default configuration	Very small percentage of users, obscure feature; affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable.	The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use.	The bug is obscure, and it is unlikely that users will work out damage potential.

(MSDN 2003)

Address each threat

Imperial College London

- Recommend a response: **META**
  - **M**itigate: make a threat harder to exploit
    - Threat: spoofing via password brute-forcing
    - Mitigations:
      - Require longer, more random passwords
      - Lock account after 3 failed attempts
      - Use biometrics instead (too expensive?)
  - **E**liminate: typically, remove the feature that was exposed to the threat
    - Longer passwords don't eliminate spoofing
    - Giving up on user accounts does (clash with business objectives?)
  - **T**ransfer: let another party assume the risk
    - We still want user accounts: "Log in with Facebook"
    - Cost: Facebook gets info about your customers
  - **A**ccept: when other options are impossible or impractical
    - Nothing can prevent a lucky hacker from guessing a password on first try
    - Important to keep track that the threat remains valid
- Cost-benefit analysis of each response depends on business objectives
- Document your response: a good way is to use a bug reporting system

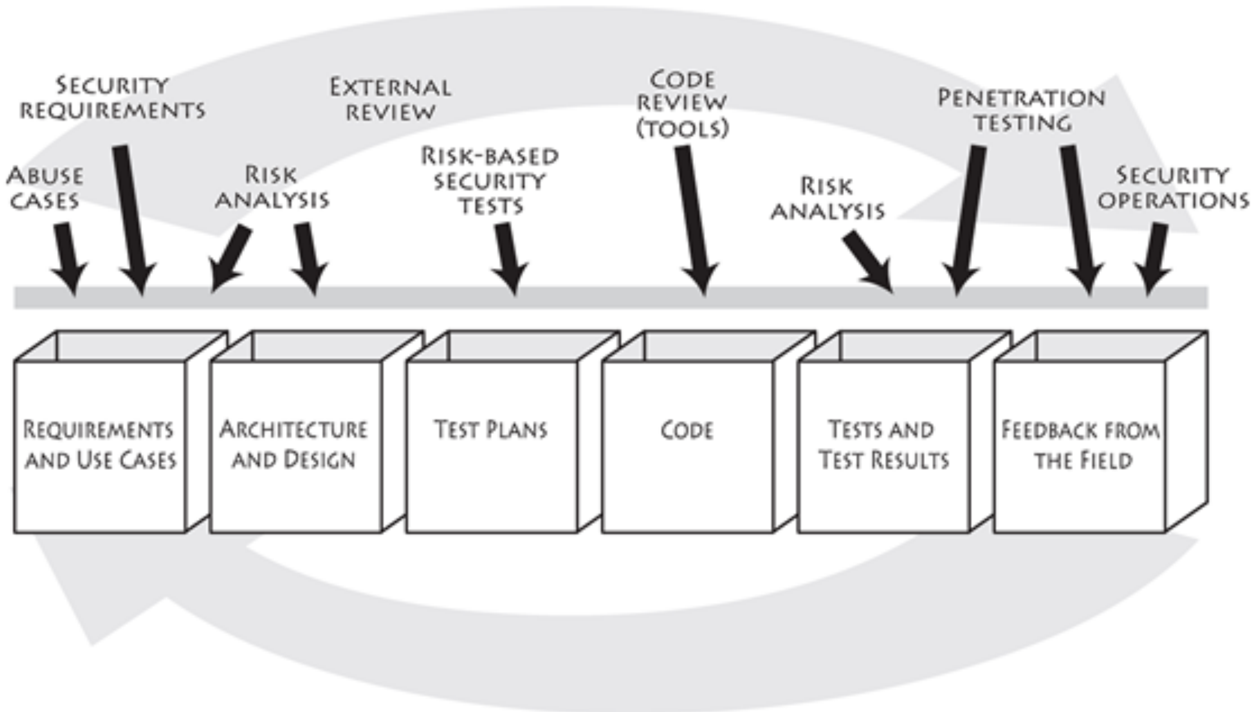
Threat modelling

Imperial College London

- Guides decision making
  - Who are the attackers, what are their goals
  - What attacks are likely to occur
  - What security assumptions does the system rely on
  - Where to invest the security budget (time, effort, money)
- Performed on model of the system
  - Free from implementation and deployment details
  - Secure design: issues can be addressed before a system is built
  - Can guide the security review of a deployed system
- More an art than a science
  - Threat modelling is a practical activity
  - Experience is key
  - There is no single way to do it right
- Three key steps
  - Model the system
  - Identify threats (STRIDE/Attack trees)
  - Evaluate and address threats (DREAD, META)

Security touchpoints

- Abuse cases
- Security requirements
- Risk analysis
  ✓ – **Threat modelling**
  – Quantitative risk assessment
- Risk-based security tests
- **Code review**
- **Penetration testing**
- Security operations



Testing

Imperial College London

- Cannot find all bugs
- Individual tests are fast to execute and cheap to produce
  - Different matter for a whole test suite
- Targeted, risk-based security tests are produced in the threat modelling phase
- Unit tests catch bugs, some bugs cause vulnerabilities
  - For example, buffer overflows
  - Although common inputs rarely exercise potential security vulnerabilities
- Fuzz testing: throw a large number of random inputs at the system, see if it breaks
  - The space of possible inputs is extremely large
    - Improvement: use mutations of common inputs
    - Even better: generate inputs based on knowledge of protocol, data format
  - Hard to tell when an input hits a vulnerability
    - Sometimes the system crashes: that's a strong hint to probe further
    - Sometimes the output looks ok, but it violates a security requirement
      - Instrument the code with additional checks that may fail during testing
      - Example: run-time check for out-of-bound array access

Static analysis

Imperial College London

- Automatically detecting security bugs is an undecidable problem
- Two main approaches
  - Some tools miss some bugs (false negatives)
    - CBMC model checker http://www.cprover.org/cbmc/
  - Some tools give spurious warnings (false positives)
    - Astree static analyzer http://www.astree.ens.fr
- Automating code review
  - Purchase commercial static anaysis tools (Coverity, HP Fortify, Checkmarx, …)
    - General purpose
    - Expensive
  - Develop tools in-house, focussed on specific problems
    - Hiring or acquisition cost
    - Facebook http://fbinfer.com tool: they bought Monoidics startup (Imperial, Queen Mary)
  - Human effort is needed to configure tools and interpret output
    - Often resistance from potential users: more bugs reported = more work, more pressure
- Static analysis techniques at Imperial
  - Abstract interpretation, Model checking, Program logics, Symbolic execution, Type systems, etc
  - Can Machine Learning play a role?

Manual code review

Imperial College London

- Can find subtle bugs not caught by other techniques
- Time consuming (=expensive)
- To our help: CWE/SANS Top 25 - list of *Most Dangerous Software Errors*
  - Based on how common, how serious, how likely to be exploited
  - Data from MITRE database of software weaknesses: http://cwe.mitre.org/top25/
  - Most recent edition dated 2011

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere