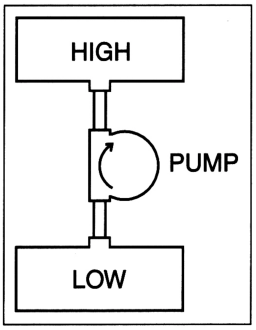
Part IA Security and Software Engineering Notes

1 Definitions

- System - A product or component [O/S comms, infrastructure, applications, internal staff, or customers/ users]
- Subject - A physical person
- Person - can also be a legal person (firm)
- Principal - A person, equipment, a role, or a complex role
- Secrecy - Mechanisms limiting the number of principles who can access information
- Privacy - Means control of your own secrets
- Confidentiality - An obligation to protect someone else's secrets
- Anonymity - Restricting access to metadata
- Integrity - An object having not been altered since the last authorised modification
- Authenticity - Either
  1. An object has integrity plus freshness
  2. You're speaking to the right principal
- Trust - A trusted system or component is one that can your security policy
- Error - A design flaw or **a deviation from an intended state**
- Failure - A nonperformance of the system
- Reliability - The probability of failure with a period of time (mean time between failure)
- Accident - An undesired, unplanned event resulting in specified kind or level of loss
- Hazard - Set of conditions on a system, plus conditions on the environment, which can lead to an accident in the event of failure
  - Failure + Hazard = Accident
- Danger - Probability of a hazard becoming an accident
- Latency - Length of hazard exposure
- Risk - Probability of an accident
  - Hazard level combined with danger and latency
- Uncertainty - Where risk is not quantifiable
- Safety - Freedom from accidents
- Security Policy - Succinct statement of protection goals
- Protection profile - Detailed statement of protection goals
- Security target - A detailed statement of protection goals applied to a particular system
- Trusted Computing Base - Set of hardware, software and procedures that can break security policy

1.1 Multilevel Secure Systems

- Enforce standard rules for different levels of classification (Confidential, Secret, Top Secret)
- Two rules
  1. Simple security policy: no read up
  2. *-policy: no write down
- Resources have classifications, principles have clearances, clearance must equal or exceed classification
- Use a pump system, a database for each level, everything gets copied up



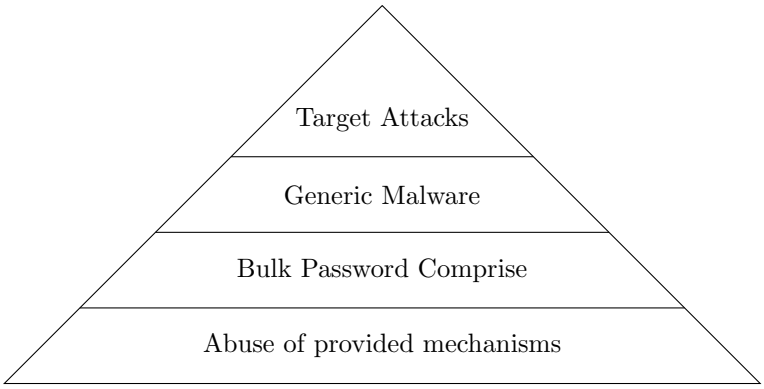
- May also want information flowing sideways - compartmentalisation
- The Biba model
  - Data may only flow down from high-integrity to low-integrity
  - Opposite of BLP
  - e.g. medical devices with 'calibrate' and 'operate' levels
- Architecture is part of security policy

2 Safety and Security Policy

- Two ways to evolve safety policies
  1. Failure modes and effects analysis (FMEA)
     - Bottom-up
     - Look at each component and list how it could fail
     - Figure out how to mitigate this (redundancy?)
  2. Fault tree analysis
     - Work back from bad outcomes to identify critical components
- Separation of duties
  - Serial - Processes must move throw multiple accountable principals
  - Parallel - Two signatures to authorise a transaction
- Role-based Access Control
  - Still a multilevel system
  - Collect users into groups and apply policies group wide
  - Group based on roles

3 Predicting User Behaviour

- Systems should try and reduce user error
- Hierarchy of harm - at each level the number of victims goes up an order of magnitude



  - Target attacks
    - Spear phishing
    - Over 50% success rate
  - Generic Malware
    - Highly distributed malware, that just sits on computer
  - Bulk password comprise
    - Passwords reused across internet
  - Abuse of standard mechanisms
    - Phising, cyber-bullying, doxxing
- Automatic behaviour leads to absent minded slips
- People are more likely to take a risk to avoid a loss instead of making a gain
- Scam mechanisms
  - Distraction
  - Social compliance - not questioning authority
  - Herd - people let down their guard when others share the same risk
  - Dishonesty - I need your help to avoid taxes
  - Kindness
  - Need and greed - know what you want and use it to manipulate you
  - Time - under time pressure
- People follow security advice that fits with their mental model of the threat
- Most people go with the defaults
- Choose the rules that matter, people only spend a certain amount of time following rules
- The right way of working should be the easiest
- People ability to spot scans is based on psychological factors
- The more complex the task the higher the risk of error

3.1 Passwords

- The cheapest way to authenticate users
- Have the following concerns
  - Will users enter passwords correctly
  - Will they remember them, will they choose weak ones or write them down?
  - Can they be tricked into revealing them?
- You can limit the number of times one can get a password wrong
- Use salt
  - Store salt with hashed password
  - Encrypt password with salt
  - Stops dictionary attacks
- Can instead use Oauth
  - But, one firm's actions have side effects
- Different websites ask for different information, allowing to chain

3.2 Security Policies

- Security policies the second core of security engineering
- Can use adversarial thinking, as part of them
- Identify Friend or Foe (IFF)
  - Obvious choice

$$F \to B : N$$
$$B \to F : \{N\}_K$$

  - The problem is that the foe can reflect back the challenge to another plane and then reply with their response
- Two-factor Authentication
  - With principals system (S), user (U), and password calculator (P)

$$S \to U : N$$
$$U \to P : N, PIN$$
$$P \to U : \{N, PIN\}_{K_P}$$

- Kerberos
  - Used to ensure that a key is fresh, communication between A and B with S being the key-granter

$$A \to S : A, B$$
$$S \to A : \{T_S, L, KAB, B, \{T_S, L, KAB, A\}_{KBS}\}_{KAS}$$
$$A \to B : \{T_S, L, KAB, A\}_{KBS}, \{A, T_A\}_{KAB}$$
$$B \to A : \{T_A + 1\}_{KAB}$$

- Europay-MasterCard-Visa (EMV)
  - With C - Card, M - Card terminal, and B - Bank
  - Both the machine and card send the transaction data to the bank so it can ensure they are the same

$$
\begin{aligned}
C &\to M : sig_B\{C, card\_data\} \\
M &\to C : N, data, Amt, PIN \text{ (if PIN used)} \\
C &\to M : \{N, date, Amt, trans\_data\}_{KCB} \\
M &\to B : \{\{N, date, Amt, tran\_data\}_{KCB}, trans\_data\}_{KMB} \\
B &\to M : \{OK\}_{KCB}
\end{aligned}
$$

- Public key cryptography
  - Public and private keys
  - Examples, Diffie-Hellman, RSA
  - Diffie-Hellman

$$
\begin{aligned}
A &\to B : g^{rA} \\
B &\to A : g^{rB} \\
A &\to B : \{M\}_{G^{rA rB}}
\end{aligned}
$$

- Public key certification
  - Physically install them on machine
  - Trust on first use - set up keys, the verify manually that you're speaking to the right person
  - Certificates, someone signs another's key
- TLS - customer C calls server S

$$
\begin{aligned}
C &\to S : C, C\#, NC \\
S &\to C : S, S\#, NS, CS \text{ (Server Certificate)} \\
C &\to S : \{K0\}_S \\
C &\to S : \text{Crypto hash of } K0,\ NC,\ NS, \text{ etc.} \\
S &\to C : \text{Crypto hash of } K0,\ NC,\ NS, \text{ etc.}
\end{aligned}
$$

- Even though a protocol is proven secure its implementation may not be
- Things usually fail when they are used in a situation they weren't tested for
- Types of bugs
  1. Arithmetic - Patriot missile - counted in 0.1, unrepresentable in binary
  2. Syntactic - Bugs that arise from specific language features
  3. Logic - Heartbleed - a buffer over-read attack
- Code injection - insert arbitary code into a system that is then run
- Software counter measures
  - Operating system - address space layout randomisation, data execution prevention
  - Strongly typed languages
  - Defensive programming - assertions
  - Secure code standards - company provided guidelines
  - Contracts - pre- and post-conditions
- Software continues to lag behind hardware's potential

4 The London Ambulance Service disaster

- Attempt to automate ambulance dispatchs in 1992
- Originally paper system
  - 999 calls written on paper tickets
  - Tickets duplicated, passed to three divisions (NW/ NE/ S)
  - Controller identifies vehicle and passed to radio operator
  - 3 minutes, a lot of staff, errors, queues, and difficult call backs
- Previous attempt failed, relation between trade union and government bad
- Consolation said would cost £1.9m and take 19 months, provided a packaged solution was found
  - Automatic vehicle location would cost even more
- Decided on a £1.5m system, with AVL and not using a packaged solution
- Of those that looked at the tender, most did not apply because the timescale was unrealistic
- Tender awarded to System Options a small company that charged much less than its competitors
- System behind schedule, only partial automation by January deadline
- Progress meeting had minuted a 6 month timescale for an 18 month project, lack of methodology, no full-time LAS users
- LAS commented that System Options was relying on 'cosy assurances' from contractors
- Server never stable in 1992, radio messaging blackspots and connection, couldn't cope with 'established working practices'
- Independent review called for volume testing, implementation strategy but was ignored
- Decided to go live on 26/10/1992
- Vicious cricle
  - System progressively lost track of vehicles
  - Exception messages scrolled off screen to be lost
  - Incidents held as allocators searched for vehicles
  - Patient call backs increased, increasing congestion
  - Slowdown and congestion lead to collapse
- System descends into chaos, switched to semi-manual later that day, fully manual on the 2 when it crashed

4.1 What went wrong

- Specification
  - LAS ignored cost and timescale advice
  - System options were insufficiently qualified and experiences and didn't know what a big system looked like
  - Specification was inflexible and incomplete, draw up without staff consolations
  - Attempted to change organisation through technical system, ignored established working practices
- Project

  - Confusion over who was managing it
  - Poor change control, not independent quality assurance, suppliers misled on progress
  - Inadequate software development tools and datacomms, with unforeseen effects
  - Poor interfaces for ambulance crews and control room
- go-live
  - Went live with serious faults
    - Slow response times
    - Workstation lockups
    - Loss of voice comms
  - Software not tested under realistic loads or as an integrated system
  - Inadequate staff training
  - No back up

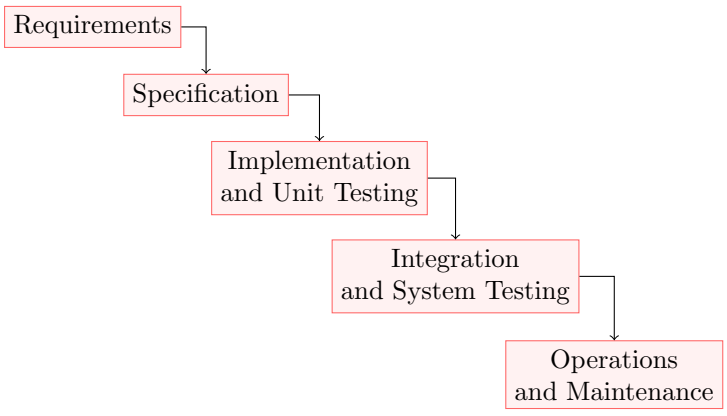
4.2 Subsequent failures

- NHS National Program for IT
- Universal Credit
- Smart Meters

5 Software Development

- Software engineering is about managing complexity from bugs and module interactions, to how users react to new functionality
- Only about 10% of the cost of software is in developing it, 90% in maintaining it
- High level languages take away a lot of the accidental complexity
- Brooks' law - adding man power to a late project makes it later
- People realised that the only way to complete a project is to split it into modules

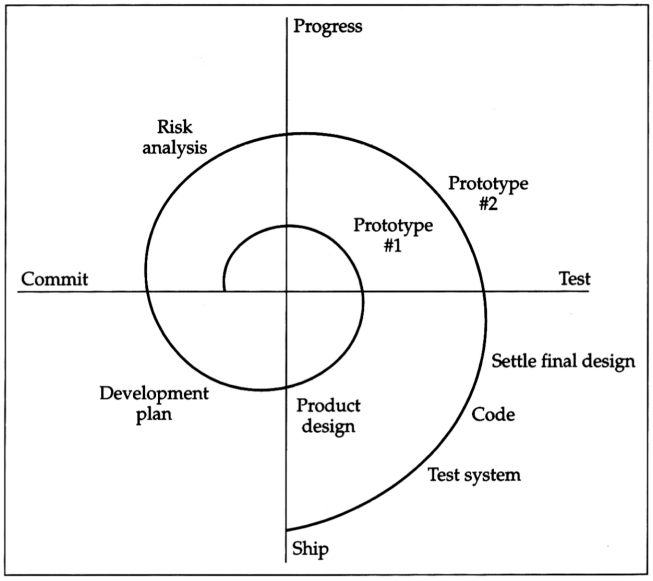
5.1 The Waterfall Model



- Requirements - written in user's language
- Specification - written in system's language

- Ensures early clarification of system goals
- Doesn't work well with iteration

5.2 The Spiral Model



- Must decide on a fixed number of iterations
- Put energy into prototyping parts not understood

5.3 Evolutionary Model

- Development cycle to add changes, test them and then deploy them
- A modern integrated development environment will have the following
  - Code and documentation version control (git)
  - Code review (gerrit)
  - Automated build (make)
  - Continuous integration (Jenkins)

5.4 Software Assurance

- Safety critical systems - failure could cause death, injury, or property damage
- Security critical systems - failure could allow leakage of confidential data
- Real time systems - Software must accomplish certain tasks on time
- Safety critical systems are often real time, the extra time pressure makes verification and testing very difficult
- Safety, security and real-time performance are system wide issues, remember users
- Usability in medical devices is very important
- Redundancy is good, but can be hard to achieve, bugs are correlated
- Building a system

1. Understand and Prioritise Hazards
2. Develop a safety case for each hazard
3. Trace hazards to source (code, hardware, etc)
4. Develop safety test plans

- Most failures don't happen during technical part of the process
- Two types of complexity
  1. **Incidental Complexity** - Keeping track of things in machine code - Solved by high level languages
  2. **Intrinsic Complexity** - Complex system with a big team - Lessened by structured development, project management tools, etc.

5.5 Tools

- Formal Methods
  - Proving programs correct
  - Not infallible
  - Can find lots of bugs, especially in small, difficult programs
- Static Analysis Tools
  - Find lots of bugs
  - For example coverity
- Programming Philosophies
  - Chief programming teams - One programmer and people to help them, team can only be so productive
  - Egoless programming - Code owned by team, but group-think entrenches bad things more
  - Literate programming - Code show be art
  - Capability maturity model - Teams should be kept together, follow five stages
    1. Initial - Starting point
    2. Repeatable - Process can be used repeatably
    3. Defined - The process is defined as a standard
    4. Managed - The process is managed according to the standard defined in the defined stage
    5. Optimised - Process management includes deliberate optimisation

5.6 Agile Development

- Split development into 2-4 week long sprints
- Have a product queue, where desired features are added
- At the beginning of each sprint choose features from the product queue and put them in the sprint queue
- Have daily scrum meetings, where only developers can speak about what they are doing
- Have a scrum master, the person in charge of the sprint
- Management doesn't change specification during the sprint
- Specification still important, an often path to failure is

Thin spread of application knowledge → Changing and conflicting requirements → break down of coordination and communication

  - Thin spread of application knowledge - People don't know everything about the business/ bank/ hospital
  - Changing and conflicting requirements - Competing products, new standards, changing environment, new customers

5.7 Management Tools

- In a specification based system there are $N(N-1)/2$ channels of communication and 2^N subgroups
- A manager's job is to plan, motivate, and control
- Gantt Charts
  - Can be hard to visualise dependencies
  - Activity block
- PERT Charts
  - A graph with dependencies
  - Directed edges are tasks, labelled with their duration
  - All tasks leaving a vertex depend on the tasks entering the vertex
  - Can find critical path (longest) and from this paths with slack in them
- To keep people motivated in groups follow the three Cs
  - Collaboration - Everyone has a specific task
  - Content - Everyone's task clearly matters
  - Choice - Everyone has a say in what they do

5.8 Testing

- Happens are many levels - UX prototyping, module testing after coding, system test after daily build, beta test, and subsequent litigation
- Regression testing - Tests if new software gives the same answers as old versions
  - Used to ensure a bug fix doesn't introduce a new error
- For 10^9 hours mtbf you must test for more than 10^9 hours
- Test in the way the system will be used, most failures occur when used outside test inputs
- Randomise tests - fuzzing
- It is hard to keep documentation in sync

5.9 Release Management

- Ensure product stable, work on recently evolved code
- Add copy protection, DRM, etc.
- Must decide whether to force upgrades or patch previous versions of the software
- In a safety critical system should test newly released software is compatible before deployment

5.10 Bugs and Vulnerabilities

- Vulnerabilities should be disclosed after a set period of time
- Bugs reported to CERT, which tells creator then publishes after a period of time
- An undisclosed bug is a zero-day bug
- Primary exploit window between discovery and patch
- If infrastructure shared then can be handled by owner
- Most industries has standards governing safety
- The world offers a hostile review (dogfood, alpha, beta)