Vulnerabilities: fundamental issues

Why do vulnerabilities exist

- Occasionally, mistakes happen in every field of engineering, but in software are so much common and pervasive
- A manufacturer producing a system with software component, must think of itself as a software company and manage software security accordingly
- Do not think that
  - every bug is obvious in hindsight. Once we see it, we wonder how the programmer could've been so stupid to not have seen it in the first place
  - We could think that all it takes is a little push against moral weakness to stop bugs from happening and that if we force programmers to be a bit more diligent, then we'll solve these cybersecurity problems
  - Ragionamenti sbagliati
- The reasons why are
  1. Vulnerabilities are an intrinsic problem of software and of cybersecurity, it is fundamentally different from every other field and cannot go away thanks to some magic technology
  2. Lack of incentives for minimizing likelihood and impact is a very complex issue
- Lack of incentives: market does not incentivize security → più sicurezza porta a costi maggiori, il produttore quindi si chiede, se lui investe di più per fornire più sicurezza, poi, quanti utenti sono disposti a pagare di più il prodotto? Non c'è molta consapevolezza sulla cybersecurity, quindi probabilmente gli utenti semplicemente smetterebbero di comprare i tuoi prodotti e si rivolgeranno ai tuoi concorrenti che gli venderanno roba meno sicura ma anche meno costosa. Oltre a tutto questo, come si fa a dire che un prodotto è più sicuro di un altro?
  - Incident costs are externalized: per quale motivo il produttore deve spendere di più per fare un prodotto più sicuro, tanto poi se ci sono dei problemi, i costi ricadono sull'utente, mica sul produttore

→ public policy is needed: example of the safety belts in cars: fino a che non c'è stata una regola imposta ai produttori che li obbligava a mettere le cinture di sicurezza nelle macchine, le cinture non c'erano, il mercato non si regolarizza da solo, bisogna imporre alcune regole (altrimenti ovviamente i produttori mica si metteranno a spendere di più per scelta loro)

- Public policy: much easier said than done
  - Establishing a practical and effective regulatory framework is very difficult
  - Say that the product should not have any vulnerability is extremely not realistic and cannot be certified → nobody can guarantee the absence of vulnerabilities, so, which guarantees should be given by the manufacturer, exactly?
  - An idea can be oblige the manufacturers to release patches, but for how long? Who will pay for that? How to update? What if the vuln is in a third party module?
  - Every system has a lot of modules by many third parties: is difficult to say who is responsible for a certain vulnerability, what if the module is not maintained? If a patch cannot be developed? Should developers of free and open source modules bear any responsibility? (Non penso che se si danno responsabilità ai proprietari di sistemi open source, poi quei sistemi rimarranno tali)
- Intrinsic problem of software and of cybersecurity
  - Non software engineer artifacts are tested for inputs representative of known operating conditions, for an untested input, the system more or less has the same behavior that it has for the tested inputs
  - For software engineer artifacts neither of the condition of the non software artifacts holds: it is an adversarial world → adversaries inject carefully selected and unexpected inputs in the system. Software is not continuous → the output of a test with a certain input tells you nothing about the system behavior with a similar but different input

→ No other technology has these vary bad features

- Adversarial does not mean hostile:
  - hostile environment: the inputs may be extreme and uncommon
  - Adversarial environment: inputs are actively and carefully selected to provoke undesired behavior, it is the worst possible input at the worst possible time

NB non-SW engineers must cope with hostile environments, SW engineers must cope with adversarial environments (they play different games)

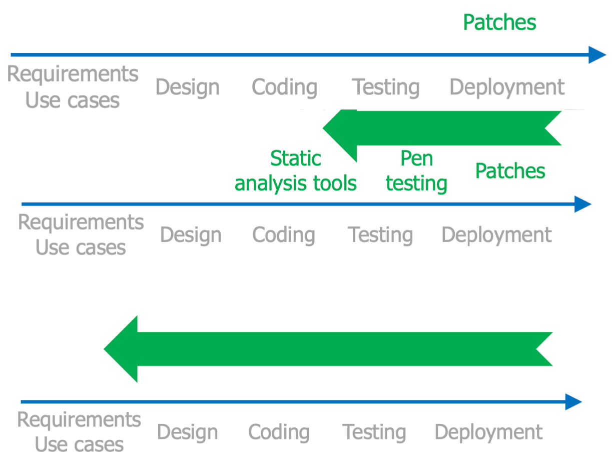
- The reason of software bugs is complexity, the only way to make software more secure is to dramatically increase the cost, change the market and reduce the amount of software being written

Vulnerabilities: fundamental issues

Cybersecurity testing

- Finora abbiamo visto che i software per loro natura devono resistere in un adversarial environment, non sono continui
- Un problema in più che bisogna affrontare è il fatto che devono soddisfare requisisti negativi (prima capiamo il testing su cose non SW e poi capiamo cosa vuol dire negative requirements)
- Testing for non-SW artifacts:
    - If we know that a system with an input of 3mW, has a signal to noise ratio at 30km of at least 46dB, for making sure that happen, we transmit 3mW and measure the signal to noise ratio at 30 km
    - In general the requirement is that a certain action can be done and the test checks that it actually can be done
    - The testing can prove that requirements are satisfied: requirements → inputs to tests → requirement satisfied
    - It there is an untested input, the behavior is not very different from the tested inputs → when the systems are delivered we are sure that they satisfy their requirements (except for occasional mistakes)
- SW (cybersecurity) testing
    - Usually there are negative requirements → a certain action cannot be done + software is not continuous
    - So the tester has to identify all the possible ways for attempting to execute the action and check them all (not feasible)
    - Example:
        - requirement: Lucifer cannot read the file x
        - Test (basic idea): all possible sizes and properties of file x, all possible sequences of inputs that include an attempt to read the file x, the cardinality of the number of test that has to be done is out of question
        - It is a fundamental problem
    - When systems are delivered we cannot be sure that they satisfy their security requirements
- Unavoidable consequence: when a software artifact is released, no one knows how many bugs it has, more testing means more confidence that is has less bugs, but nobody can tell that there is no bug or how many bugs there are
- Cybersecurity guarantees: what we would need is a proof of a near proof that the system will do what is should and the system will not do what is should not → not feasible due to the fact that is in an adversarial environment, software is not continuous and it has negative requirements
    - The only guarantee that we can achieve is some degree of confidence (assurance) che tutto il sistema si comporti come dovrebbe, non c'è però un modo per misurare o quantificare questo grado di fiducia (soggettivo)
- Assurance:
    - There are many complementary techniques ("shifting left")
        - Programming language and methodology, development process, testing methodology and effort, penetration tests…
        - High assurance systems are the one with strong and proven usage of such techniques

Shifting left



- Software development life cycle (SDLC) → Manufacturer behavior can be:
    1. Do nothing
    2. Tackle security issues after deployment
    3. Evolve to tackle them earlier (shift left)
- Shifting left: Do nothing - Patch - Patch management system - Bug bounty, pen testing, security testing - Static analysis tools when coding - Security training to programmers - Think of abuse cases and develop specific security tests - Think about security before coding: security requirements, architecture review and so on…
  (Varie (chiamiamole) tecniche per fare ciò)

NB we don't know how to make software secure, but we do know how to make software more secure, it is a slow evolution, a necessary condition is strong incentives

Lo shifting left praticamente consiste essenzialmente nel affrontare e risolvere alcuni problemi in modo precoce, nella fase di sviluppo e progettazione del sistema → lo scopo è quello di integrare le considerazioni sulla sicurezza già dall'inizio