Chapter 1: Security Concepts and Principles

SYSC 4810: Introduction to Network and Software Security

Prof. Hala Assal

What is Computer Security?

The combined art, science and engineering practice of protecting computer-related assets from unauthorized actions and their consequences, either by preventing such actions or detecting and then recovering from them.

Security Properties

Confidentiality
Non-public information is accessible only to authorized parties

Integrity
The property of assets is unaltered except by authorized parties

Authorization
The property of computing resources is accessible only by authorized entities

Authentication
Assurance that a principal, data, or software is genuine relate to expectations arising from appearance or context

Security Properties

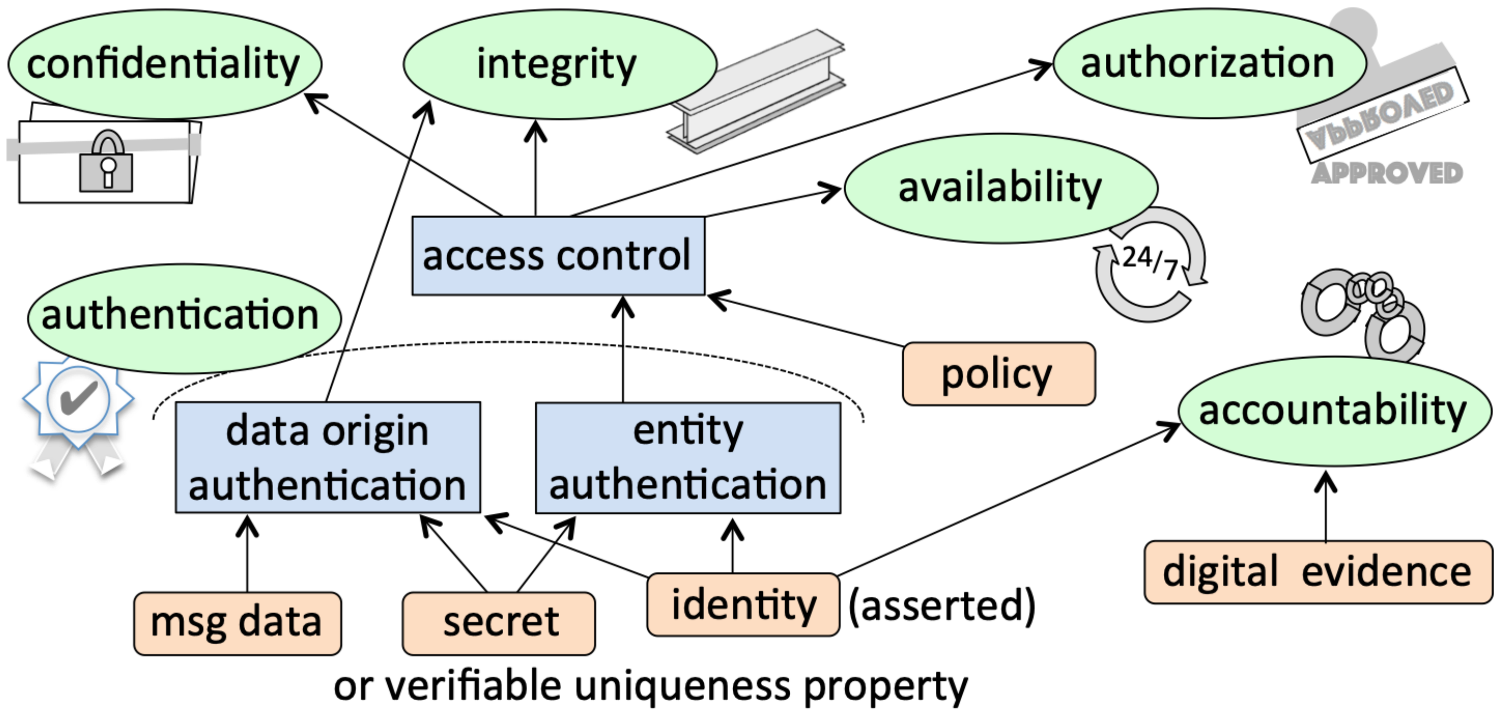
Availability

The property of assets remains accessible for authorized use

Accountability

The ability to identify principals responsible for past actions

Security Goals and Mechanisms



Terminology

- Trusted vs Trustworthy
  - Has vs deserves our confidence
- Confidentiality *vs* Privacy *vs* Anonymity
  - Secrecy
  - PII
  - Actions

Security Policy

"Specifies the design intent of a system's rules and practices—what is, and is not (supposed to be) allowed"

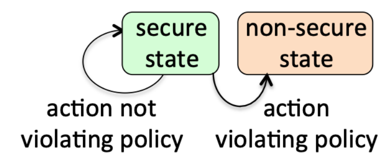
- Dictates or is derived from "Security Requirements"
- Authorizes system states
- Allows for measuring violations

Security Attack

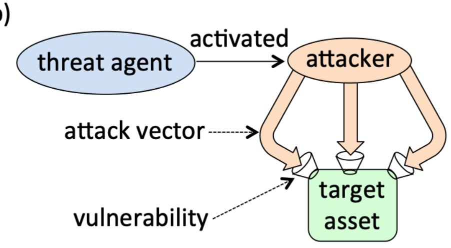
“The deliberate execution of one or more steps intended to cause a security violation, such as unauthorized control of a client device”

- Attacks exploit **vulnerabilities**
  - Design/implementation flaws
  - Deployment/configuration issues
    - Lack of physical isolation, ongoing use of known default passwords, debugging interfaces left enabled
- Adversary vs Attacker
  - Adversary: the *threat agent* behind a potential attack
  - Attacker: the adversary that has activated the threat into an attack

a)



b)



Threat

"Any combination of circumstances and entities that might harm assets, or cause security violations"

- Credible threat: means and intent
- Computer Security aims to protect assets by:
  - Identifying and eliminating vulnerabilities thus disabling attack vectors
    - Specific methods, sequence of steps, by which attacks are carried out
- Threat agents and attack vectors raise the question: *secure against whom, from what types of attacks?*

Controls (countermeasures)

- Needed to support and enforce security policies
- Include:
  - Operational and management processes
  - OS enforcement by software monitors
  - Access control measures
  - Specialized devices, software techniques, algorithms and/or protocols

Attack | Security violation
Attack vector | Threat
Countermeasure | Vulnerability

E.g., a House Security Policy

1) No one is allowed in the house unless accompanied by a family member.

2) Only family members are authorized to remove physical objects from the house.

- Having an unaccompanied stranger in the house is a **…?…**
- An unlocked back door is a **…?…**
- A stranger entering through such a door, and removing a television, amounts to an **…?…**
- Entry through the unlocked door is **…?…**
- A **…?…** here is the existence of an individual motivated to profit by stealing an asset and selling it for cash.

E.g., a House Security Policy

1) No one is allowed in the house unless accompanied by a family member.

2) Only family members are authorized to remove physical objects from the house.

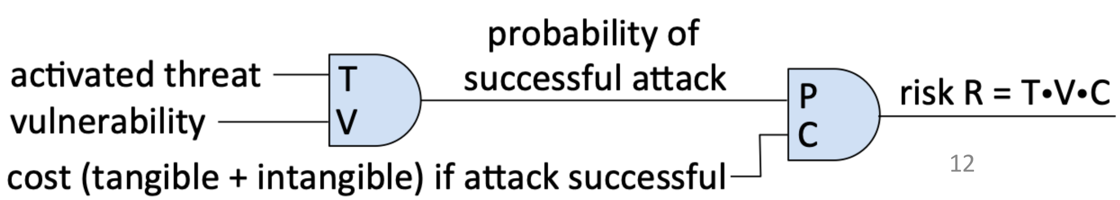
- An unaccompanied stranger in the house is a **security violation.**
- An unlocked back door is a **vulnerability.**
- A stranger entering through such a door, and removing a television, amounts to an **attack.**
- Entry through the unlocked door is **an attack vector.**
- A **threat** here is the existence of an individual motivated to profit by stealing an asset and selling it for cash.

- What **countermeasures** can we have in place to enforce the security policy?

Risk (**R=P.C**)

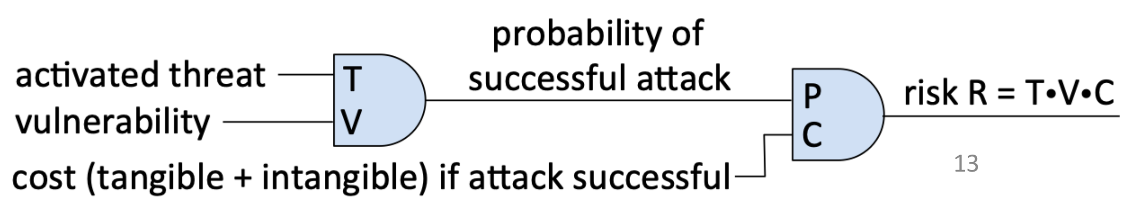
"The expected loss due to harmful future events, relative to an implied set of assets and over a fixed time period"

- Depends on:
  - Threat agents, attack probability, and expected losses
- Assets e.g.,
  - Software applications, files, databases, client machines, servers and network devices



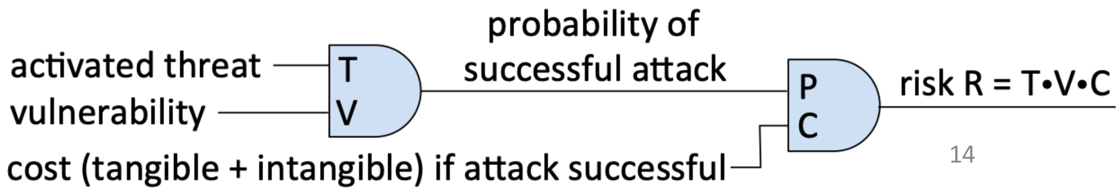
Risk Assessment

- **While calculating risk, we ask:**
  - What assets are most valuable, and what are their values?
  - What system vulnerabilities exist?
  - What are the relevant threat agents and attack vectors?
  - What are the associated estimates of attack probabilities, or frequencies?



Risk Assessment Challenges

- Incomplete knowledge of vulnerabilities
  - Rapid technology evolution
- The difficulty of quantifying the value of intangible assets
  - Strategic information, corporate reputation, etc.
- Incomplete knowledge of adversary classes
  - Actions of unknown intelligent human attackers cannot be accurately predicted
  - Their existence, motivation, and capabilities evolve, especially for *targeted attacks*.



Qualitative Risk Assessment

C (cost or impact)	P (probability)				
	V.LOW	LOW	MODERATE	HIGH	V.HIGH
V.LOW (negligible)	1	1	1	1	1
LOW (limited)	1	2	2	2	2
MODERATE (serious)	1	2	3	3	3
HIGH (severe or catastrophic)	2	2	3	4	4
V.HIGH (multiply catastrophic)	2	3	4	5	5

Table 1.1: Risk Rating Matrix. Entries give coded risk level 1 to 5 (V.LOW to V.HIGH) as a qualitative alternative to equation (1.2). V. denotes VERY; C is the anticipated adverse effect (level of impact) of a successful attack; P is the probability that an attack both occurs (a threat is activated) and successfully exploits a vulnerability.

Cost-Benefit Analysis

- Helps with deciding budgets
  - e.g.: Cost-benefit analysis of password expiration policies
- Risk management: *(technical + business)*
  - Risk mitigation
    - By technical or procedural countermeasures
  - eliminating risk by decommissioning the system
  - transferring risk to third parties, through insurance
  - accepting risk in the hope that doing so is less costly than the above two points

Adversary Modeling

- Objectives:
  - target assets/systems
- Methods:
  - attack techniques/types
- Capabilities:
  - computing resources, funding sources, skills, knowledge, personnel, opportunity
- Motivation:
  - Financial reward, hurting reputation, ego, criminal, political
- Outsider vs Insider:
  - Outsider: an attack launched without any prior special access to the target network
  - Insider: originates from parties having some starting advantage

Adversary Groups

- *By names* →
- By capabilities
- By aim
- Etc…

	Named Groups of Adversaries
1	foreign intelligence (including government-funded agencies)
2	cyber-terrorists or politically-motivated adversaries
3	industrial espionage agents (perhaps funded by competitors)
4	organized crime (groups)
5	lesser criminals and *crackers*† (i.e., individuals who break into computers)
6	malicious *insiders* (including disgruntled employees)
7	non-malicious employees (often security-unaware)

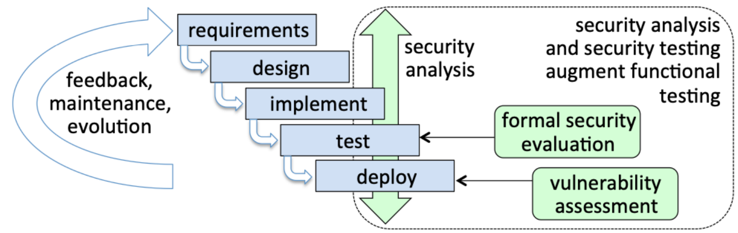
Security Evaluation

- Certification
  - Third party lab reviewing (considerable cost and time)
  - Re-certification is required once even the smallest changes are made!
- Self-assessments
  - Penetration testing (pen testing) to find vulnerabilities in deployed products
  - Interactive and automated toolsets run attack suites that pursue known design, implementation, and configuration errors compiled from previous experience.

Pen Testing

- Traditionally black-box
- White-box pen testing
  - Increases the chances of finding vulnerabilities
  - Allows tighter integration with overall security analysis
- Tests carried out by product vendors prior to product release cannot find all issues
  - e.g., those arising from customer-specific configuration choices and deployment environments

Security Analysis



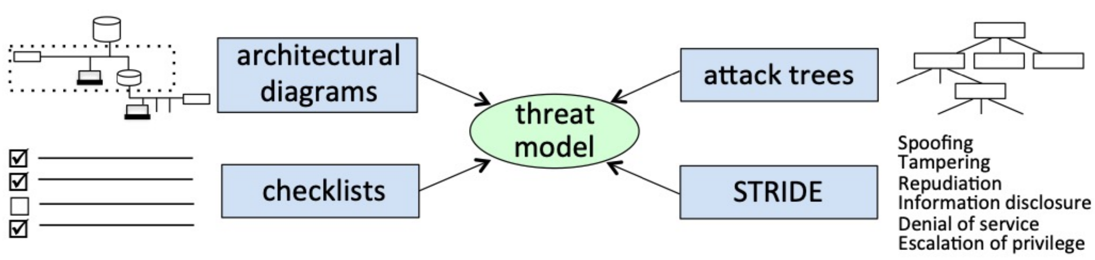
- Aims to:
    - Identify vulnerabilities related to design, and overlooked threats
    - Suggest ways to improve defenses when weaknesses are found
- Analysis ideally begins early in a product's lifecycle, and continues in parallel with design and implementation
- Manual source code review can uncover vulnerabilities not apparent through black-box testing alone
- Analysis should trace how existing defences address identified threats
    - … and notes threats that remain unmitigated.
- Vulnerability assessment
    - The process of identifying weaknesses in deployed systems, including by pen testing

Security Model

- Relates system components to parts of a security policy
- Model can be:
  - Explored to increase confidence that system requirements are met
  - Designed prior to defining policies

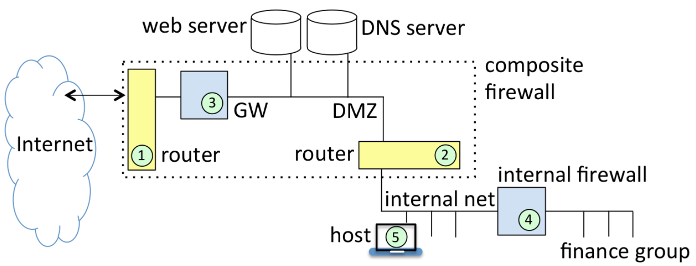
Threat Model

- Identifies threats, threat agents, and attack vectors considered in scope
  - Either known from the past and/or anticipated.
- Threat model also defines out-of-scope elements.
- Accounts for adversary modeling
- Should identify and consider all assumptions made about the target system, environment, and attackers.



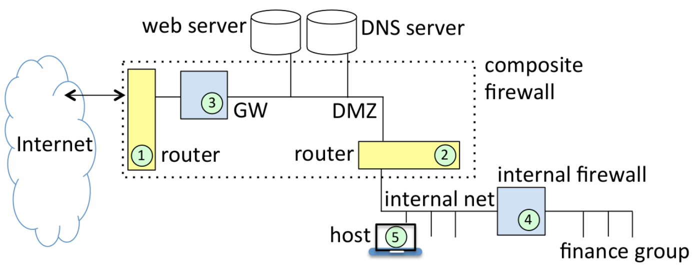
Threat Modeling: Diagram-Driven

- A visual approach to threat modeling.
- Starts with an architectural representation of the system to be built or analyzed.
- Steps:
  - Draw a diagram showing system components and network links.
  - Identify and mark system gateways where system controls restrict or filter communications.
  - Use these to delimit what might informally be called trust domains.



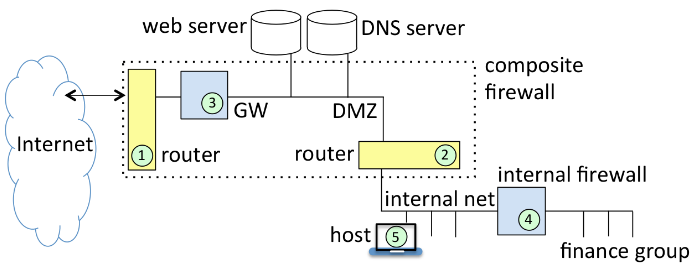
Threat Modeling: Diagram-Driven

- E.g., if users log in to a server, draw a rectangle around the server to denote that this area has different trust assumptions
  - users within this boundary must e.g., be authenticated.
  - or data within this boundary has passed through a filter.
- Now:
  - Ask how your trust assumptions, or expectations of who controls what, might be violated.
  - Focus on each component, link and domain in turn.
  - Ask: *Where can bad things happen? How?*



Threat Modeling: Diagram-Driven

- Add more structure and focus to this process by turning the architectural diagram into a data flow diagram
  - trace the flow of data through the system for a simple task, transaction, or service.
  - Examining this, again ask: *"What could go wrong?"*
- Then consider more complex tasks, and eventually all representative tasks.



Threat Modeling: Diagram-Driven

- Consider user workflow
  - trace through user actions from the time a task begins until it ends.
  - Begin with common tasks. Move to less frequent tasks
    - e.g., account creation or registration (de-registration), installing, configuring and upgrading software (also abandoning, uninstalling).
- Consider full lifecycles of data, software, accounts.
- Revisit your diagram,
  - and highlight where sensitive data files are stored on servers, user devices?
- Double-check that all authorized access paths to this data are shown.
  - Are there other possibilities, e.g., access from non-standard paths? How about from back-up media, or cloud-storage?
- Revisit your diagram:
  - Now add in the locations of all authorized users, and the communications paths they are expected to use.
- Are any paths missing—how about users logging in by VPN from home offices?

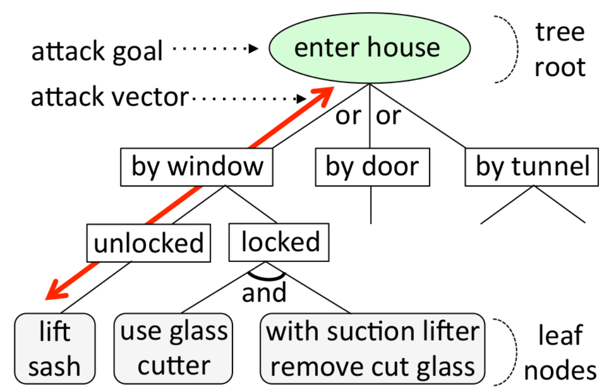
Threat Modeling: Diagram-Driven

- Are all communications links shown, both wireline and wireless?
  - Might an authorized remote user gain access through a Wi-Fi link in a cafe, hotel or airport - could that result in a middle-person scenario?
  - If someone nearby has configured a laptop as a rogue wireless access point that accepts and then relays communications, serving as a proxy to the expected access point?
  - Might attackers access or alter data that is sent over any of these links?
- Revisit your diagram again:
  - Who installs new hardware, or maintains hardware?
  - Do consultants or custodial staff have intermittent access to offices?
- The diagram is just a starting point, to focus attention on something concrete.
- The diagram must be looked at in different ways, expanded, or refined to lower levels of detail.
- The objective is to encourage semi-structured brainstorming, get a stream of questions flowing, and stimulate free thought about possible threats and attack vectors

That's how threat modeling begins, an open-ended task...

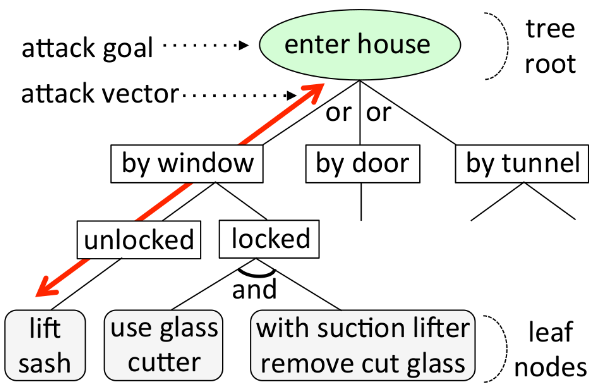
Threat Modeling: Attack Trees

- Good to identify attack vectors.
- A tree starts with a root node at the top, labeled with an overall attack goal (e.g., enter a house).
- Lower nodes break out alternative ways to reach their parent's goal
  - E.g., enter through a window, through a door, tunnel into the basement.
- Each may similarly be broken down further
  - E.g., open an unlocked window, break a locked window.
- Each internal node is the root of a sub-tree whose children specify ways of reaching it.
- A path connecting a leaf node to the root lists the steps composing one full attack.
  - Cf., attack vectors



Threat Modeling: Attack Trees

- Multiple children of a node are distinct alternatives
- A subset of nodes at a given level can be marked as an AND set
  - i.e., all are jointly necessary to meet the parent goal.
- Nodes can be annotated with detail
  - e.g., a step is infeasible,
  - Could also refer to costs or other measures.
- The attack information can often suggest natural classifications of attack vectors into known attack categories.



Threat Modeling: Attack Trees

- Attack trees output an extensive list of possible attacks *(but usually incomplete)*
- Attack paths can be examined to determine which ones pose a risk in the real system.
  - If the circumstances detailed by a node are infeasible in the target system, the path is marked invalid → helps to focus on relevant threats
- Note: an attacker need only find **one way** to break into a system,
  - while the defender must defend against **all viable** attacks.
- Attack trees can help forming security policies
- Attack vectors identified help determine the types of defensive measures.
- Attack trees can be used to prioritize vectors as high or low
  - e.g., based on their ease, and relevant classes of adversary.

Threat Modeling: Attack Trees

- The attack tree methodology encourages directed brainstorming.
  - Reducing ad-hoc-ness
- The process:
  - Benefits from a creative mind.
  - Requires a skill that improves with experience.
  - Is also best used iteratively, with a tree extended as needed
- Attack trees motivate security architects to *“think like attackers”*, to better defend against them.

Threat Modeling: Checklists

- Consulting fixed attack checklists
  - drawn up over time from past experience by larger communities, and accompanied by varying levels of supporting detail.
- Advantages: Extensive checklists exist!
  - their thorough nature can help ensure that well-known threats are not over-looked by ad hoc processes
  - may require less experience or provide better learning opportunities.
- Disadvantages: such pre-constructed generic lists contain known attacks in generalized terms
  - No unique details/assumptions of the target system and environment in question
  - they may themselves overlook threats relevant to particular environments and designs
  - long checklists are tedious, replacing a security analyst's creativity with boredom
- Checklists are best used as a complementary tool to other threat modeling schemes

Threat Modeling: STRIDE

Spoofing: attempts to impersonate a thing (e.g., web site), or an entity (e.g., user).

Tampering: unauthorized altering, e.g., of code, stored data, transmitted packets

Repudiation: denying responsibility or past actions

Information disclosure: unauthorized release of data

Denial of service: impacting availability/quality of services through malicious actions

Escalation of privilege: obtaining privileges to access resources

- The idea is to augment the diagram-driven approach by asking:
  - *Where can things break?*
- STRIDE thus stimulates open-ended thoughts, guided by six keywords.

Model—Reality Gaps

- How accurate is threat modeling?
  - Does it focus on the wrong threats?
  - Over abstraction and simplification
    - Devil is in the details
- Hotel safebox example
  - Who did you implicitly trust?
- Implicit trust within threat modeling
  - Failure to record assumptions explicitly
  - Misplaced trust
- How accurate can it get?
- How often does it need updating?
  - Again: Rapid technology evolution

Examples of Failed Threat Modeling

- Disabling online bank transfers to protect cleaning compromised accounts
  - Adversary purchases its own product with funds from the compromised account
- Using a list of one-time passwords to exhaust password leaks
  - A phishing website asks for a few passwords from the list
- Traditional network perimeter defenses
  - BYOD, USB tokens scattered in a parking lot, or s/w installations need not damage the firewall to get in
- Google Chrome's "Secure" label
  - Malicious sites with valid certificates will be labeled as so

Internet Threat Modeling

- Two fundamental assumptions:
  1. End-points, e.g., client and server machine, are trustworthy
  2. The communications link is under attacker control (subject to eavesdropping, message modification, message injection).
- Follows the historic cryptographer's model
  - securing data transmitted over unsecured channels.
- Assumption (1) often fails in today's Internet
  - E.g., malware and keyloggers

Practical Aspects

TESTING IS NECESSARILY INCOMPLETE

SECURITY IS UNOBSERVABLE

ASSURANCE IS DIFFICULT

Testing is incomplete

- How do we test that the protection measures work and that the system is "secure"?
- What is the definition of “secure”?
  - Follows Security Policies?
  - Are the policies enough?
  - Are the adversary and threat models captured properly to answer the above two questions?
  - Any implicit or inaccurate assumptions?
- How to test if security requirements have been met?
  - Remains an open question to date!
- Tests can be done using checklists, known attacks, common flaws, etc to see if a system successfully withstands them.
- Can we test for unaddressed attacks not yet foreseen or invented?
- Assurance is thus incomplete, and often limited to well-defined scopes.

Security is Unobservable

- Security testing would ideally confirm the absence of vulnerabilities.
- Naturally, a **negative goal!**
- This is not possible.
- If we never saw a black swan, can we **prove** all swans are white?
- The universe of potential exploits is unknown.
- A system's security properties are thus difficult to predict, measure, or see
- We cannot observe security itself or demonstrate it, albeit on observing undesirable outcomes we know it is missing
- The security of a computer system is not a testable feature, but rather is said (unhelpfully) to be emergent—resulting from complex interaction of elements that compose an entire system.

Assurance is difficult

- Evaluation criteria are altered by experience
  - even thorough security testing cannot provide 100% guarantees.
- We seek to iteratively improve security policies
  - Thus renew our confidence that protections in place meet security policy and/or requirements.
- Assurance of this results from:
  - sound design practices
  - testing for common flaws and known attacks using available tools
  - formal modeling of components where suitable
  - ad hoc analysis
  - heavy reliance on experience.

- **The best lessons often come from attacks and mistakes.**

Security Design Principles

P1: Simplicity-and-necessity

- Minimal installs, minimal functionality
- Minimize attack surfaces

P2: Safe-defaults

- Deny access by default
- Fail safe
- Strong default passwords
- HTTPS by design

P3: Open-design

- Security by obscurity 🚫

P4: Complete-mediation

- Authentication and authorization

Security Design Principles

P5: Isolated-compartments

- E.g., system memory isolation (e.g., Android)
- Prevent privilege escalation

P6: Least-privilege

- E.g., do not distribute super accounts

P7: Modular-design

- Cf. Tanenbaum vs Torvalds
- Favour Object-oriented and fine-grained designs

P8: Small-trusted-bases

- E.g., microkernel architectures, crypto separates algorithms from secrets

Security Design Principles

P9: Time-tested tools

- Systems that stood the test of time are more conclusive

P10: Least surprise

- Align designs with users' mental models
- Tailor to the experience of target users
- Designs suited for trained experts but unintuitive or triggering mistakes by typical end users
- Simpler, easier-to-use, usable mechanisms yield fewer surprises

P11: User-by-in

- Design systems that encourage users to behave securely

P12: Sufficient-work-factor

- The cost to defeat a system is larger than the expected adversary's capabilities

Security Design Principles

P13: Defence-in-depth

- Place a defence mechanism at each stage where one can be placed
- Avoid single point of failures
- And defence in breadth!

P14: Evidence-production

- Logging and forensics

P15: Data-type-verification

- Sanitize any input, no matter where it came from

P16: Remnant-removal

- E.g., clear memory after program termination

Security Design Principles

P17: Trust-anchor-justification

- Trust anchors are dangerous!
- Ensure their trustworthiness

P18: Independent-confirmation

- E.g., keys and software hash confirmations

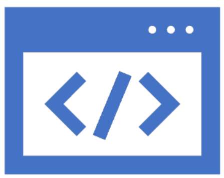
P19: Request-response-integrity

- Verify that responses match requests
- E.g.,: a certificate request expects in response a certificate for that subject.

P20: Reluctant-allocation

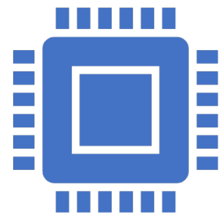
- … of resources; e.g., to deter DoS
- Place a higher burden of proof of identity or authority on agents that initiate a communication or interaction.

Higher-Level Principles



HP1: Security-by-design

Do not make security an independent added layer at the end.



HP2: Design-for-evolution

Algorithm agility

Backward compatibility

Efficient and secure system updates

Why Security is Hard!

- Intelligent, adaptive adversary
  - and is often economically motivated.
- No rulebook
  - while defenders typically follow protocols, standards and customs.
- Defender-attacker asymmetry
  - attackers need only one weakness; defenders must protect all.
- Scale of attack
  - Facilitated by the Internet’s easily reproduced and amplified communications.

Why Security is Hard!

- Universal connectivity
  - … and low traceability/physical risk.
- Pace of technology evolution
  - continuous software upgrades and patches.
- Software complexity
  - and complexity is the enemy of security.
- Developer training and tools
  - many developers have no security training.

Why Security is Hard!

- Interoperability and backwards compatibility
  - interoperability requirements complicates deploying security upgrades
- Market economics and stakeholders
  - stakeholders who can improve security may not be those gaining its benefit.
- Features beat security
  - little market exists for simpler products with reduced functionality.
- Low cost beats quality
  - low-cost low-security wins because high quality software is indistinguishable from low (other than costing more)
  - and when software sold has no liability for consequential damages.

Why Security is Hard!

- Missing context of danger and losses
  - consequences of security breaches are often not linkable to the cause.
- Managing secrets is difficult
  - … due to the nature of software systems and human factors.
- User non-compliance (human factors)
  - users undermine computer security mechanisms that has no visible benefits.
  - (in contrast: physical door locks are also inconvenient, but benefits are understood).
- Error-inducing design (human factors)
  - it is hard to design security mechanisms whose interfaces are intuitive, distinguishable from attackers' interfaces, induce the desired human actions, and resist social engineering.

Why Security is Hard!

- Non-expert users (human factors)
  - Users are non-experts without formal training or technical background.
- Security not designed in (originally)
  - retro-fitting it in the Internet as an add-on feature is impossible without major redesign.
- Introducing new exposures
  - the deployment of a protection mechanism may itself introduce new vulnerabilities or attack vectors.
- Government obstacles
  - government desire for access to data and communications (e.g., to monitor criminals, or spy on citizens and other countries) hinders protection practices such as strong encryption by default.