# Today

Security vs risk management

Adversarial thinking

Abstraction and its problems

Trust and TCBs

# Risk management

## Computers not the only risky systems!

- reliability
- safety
- fraud detection
- epidemiology

## Q: what do these have in common?



**A: a couple of things**

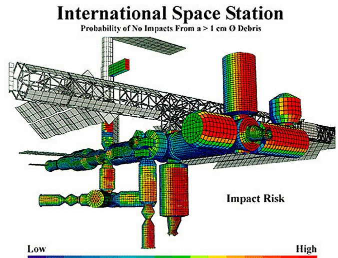
- like security: hidden problems that come to light
- unlike security: quantitative analysis

# Stochastic threats

**Reliability:** probability of failure / time between failures

**Safety:** probability of failures causing safety incident

**Epidemiology:** probability of infection after exposure




## Risk equation:

$$R = P \times C = T \times V \times C$$

## Q: On what do these probabilities depend?



We often assume that different risks are ____________________. This can be quite reasonable in the case of safety engineering, reliability engineering, etc.. If rust can rust, ______________. How much? _________________________ If a virus can infect you, ______________.

Although, there is one wrinkle in the case of epidemiology: as we've all seen, it's not just about how the _____________ will behave, how the ___________________ will behave is also pretty important!

# Know your enemy

## Classical risk management

- an impersonal force of nature

## Computer security (and crime, and geopolitics…)

- defending against **people** taking **intentional** actions
- not just a force, an **adversary**, an **attacker**



Crime isn't just a matter of means and opportunity: it's also a question of ______________ (as well as ______________, ______________ and _____________________).

The presence of an adversary (or adversaries) is what makes security different from mere risk management.

# Adversarial thinking

**The attacker:**

## a directed, strategic, *adaptive* adversary



__________________ *wants* something

__________________ makes *choices* and *plans* to enhance effectiveness

A flood or a virus doesn't choose where or when to strike

Example: lighting and bird strikes

__________________ will change attacks as you change defences

# Thinking about adversaries

## Adversaries vary in their:

- Objectives
- Capabilities
- Methods
- Insider access
- Support



Objectives: different adversaries want different things! Money, revenge, policy change or just "for the lulz".

Capabilities: some adversaries are technically very savvy and capable, others are not. Capabilities can also include non-technical capabilities: an adversary who can ______________________ opens up possibilities that strictly technical adversaries don't have. "Unsophisticated" doesn't mean "__________", though!

Methods: not just what they're capable of, but what they like to do and even what they're willing to do. Different adversaries have different approaches that they take, and some are willing to use approaches that other's aren't.

Insider access: we'll talk more about this in a moment, but a disguntled insider (or someone who can find/cultivate one) is actually a very powerful adversary.

Support: some adversaries are on their own, poking at servers in their free time, whereas others are funded to develop campaigns full-time with teams around them to support their activities. Defending against one is very different from defending against the other.

# Adversary models

## Can do some formal modeling

e.g., the *Dolev-Yao* attacker is very important in network security

## Informal shorthands often more immediately useful

# Informal adversary models

| | |
|---|---|
| Accidental | Intelligence service |
| APT | Military |
| Competitor | Lookie-loo |
| Hacktivist | Organized crime |
| Honest-but-curious | Scammer |
| Insider | Script kiddie |



| | |
|---|---|
| Accidental | Violates security policy without meaning to |
| APT | Well-resourced, operate with impunity |
| Competitor | Industrial espionage |
| Hacktivist | Social or political motivation |
| Honest-but-curious | Executes protocols faithfully but sneaks a peek |
| Insider | Disgruntled employee, whistleblower, etc. |
| Intelligence service | Well-resourced, connected to non-cyber assets |
| Lookie-loo | Motivated by curiosity |
| Military | Connected to physical-world objectives |
| Organized crime | Financial incentive, well-organized markets |
| Scammer | Financial incentive, low effort |
| Script kiddie | Want to see what they can do |

# Abstraction

### What is abstraction?

### Why is it helpful?

### How is it deceptive?

"Towards a New Model of Abstraction in the Engineering of Software", G Kiczales, *IMSA'92: Proceedings of the 1992 Workshop on Reflection and Meta-level Architectures*, 1992.
"The Law of Leaky Abstractions", J Spolsky, *Joel on Software*, 2002.



You've been thinking in a structured way about abstraction since your ______________________ __________, and informally for long before that! Abstraction is useful; in some ways, it's the core of what all engineers do.

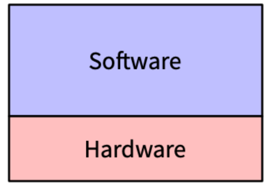
Abstraction is useful, as it allows us to ______________ some aspects of a problem while we _____________ on others — we can't ___________________________________! For example, it would be much harder to write Python code that translates objects to JSON respresentations if we had to be concerned with the implementation details of how, say, a hash map is implemented (what Marsenne prime is being used?), or what the virtual address of an object is, or how that virtual address is translated to a physical address, or which L2 cache line it's occupying!

On the other hand, abstractions are _____________. A remote method invocation interface may hide all of the details of network configuration and method enumeration, but if the network goes down, it can't hide that problem (or at least not well!). Complex systems require thinking __________________________________; if you aren't, you can be sure that your attackers are!

# Abstraction layers

## Common model of a computing system:

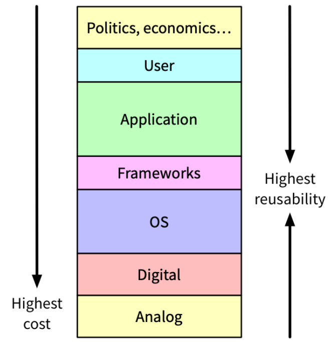
- attacker can attack the software
- attacker can attack the hardware



# More abstraction layers!

**More realistic model of a computer system:**

- attacks can come at *any* layer
- defence must happen at *every* layer
- attacks can be as hidden as implementation details






The real world is complicated. We have lots of abstractions that go into the making of a computer system, and all of them leak! None of them fully hide the details of the layers below, and none are immune from the influence of the layers that sit on top of them. Security is ______________ and ______________.

Critically for security, the attacker often gets to meet you on a __________________________. If one abstraction layer of your system defends effectively against an attacker, they can often come at layers ____________ or ____________ your work. A bank's smart card can perform a lot of cryptographic operations to help safeguard your information, but those aren't enough by themselves. In a ____________ layer, an adversary can attempt to exploit ________________ ____________________ of the card itself to learn secret information like cryptographic keys. At a _____________ layer, if the adversary can gather card details including the CVV2 code via a skimmer or by fooling the cardholder, all the side-channel security in the world can't protect you. Thus, your defences are often only as strong as _______________________. Example: Bunker Buster, *The Daily WTF*

Technical people like engineers often don't like to think about the highest-level abstractions on this chart, but they are real! The best cryptography and other technical measures can be easily subverted if you can trick users into misusing systems, or if the economic incentives of a larger sociopolitical system reward bad behaviour.

# Really? Users?

## Security is a *human* discipline

- attacker motivations
- defender motivations
- insider motivations

*Office Space (1999)*



Insiders can ________________ malicious

# Secondary goal

> *Security is usually a **secondary goal**. People do not generally sit down at their computers wanting to manage their security; rather, they want to send email, browse web pages, or download software, and **they want security in place to protect them** while they do those things. It is easy for people to put off learning about security, or to optimistically assume that their security is working, while they focus on their primary goals. Designers of user interfaces for security should not assume that users will be motivated to read manuals or to go looking for security controls that are designed to be unobtrusive.*

*Usability of Security: A Case Study*, Whitten and Tygar, CMU-CS-98-155



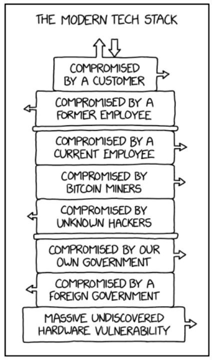
This quote is from Dr Whitten's 1999 PhD thesis (which came out during the same year as Office Space!).

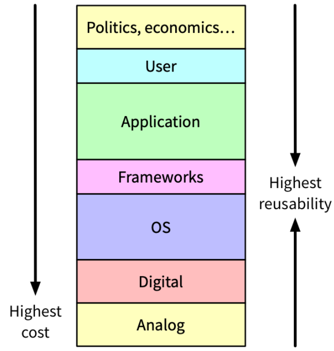
Don't make users' lives ______________________! You may turn them into ______________________.

# Trust and TCBs

**What is trust?**

**"Trusted" vs "Trustworthy"**






Trust is typically a word that brings ______________________, but not in this course!

Do you trust your bank? _______________ You actually trust a combination of your bank teller, double-entry bookkeeping, security cameras, time vaults, police and security guards, but also — much more than most people think about — the Canada Deposit Insurance Corporation.

Someone that you might really trust is a __________________. If you meet with a _________, you will explain your clever idea for a ___________________________ but they will not ____________________________. You will have no guarantee that they won't just _______________________... now *that* is trust. Do you feel __________ _______ about that?

We should build systems that are _______________ without assuming that they are ___________.

# One definition of "trusted"

> *A trusted system is one whose failure can break the security policy*

*Anderson, Security Engineering*

In this view:

**Something you *have* to trust, not *want* to trust**



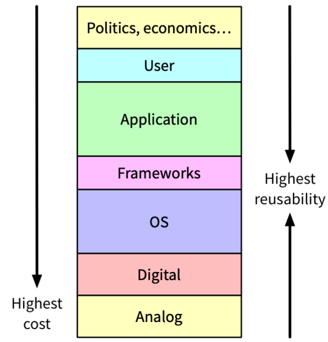
**Or: "one that can get you fired"**

**Or: "one that you can't really validate"**

# TCB: Trusted Computing Base

**Everything you have to trust**

**Goal: minimize!**






A *trusted computing base* is everything in a system that you are trusting, i.e., everything you are depending on in order for your part of a system to work correctly.

Attacks against different layers have different costs and different levels of applicability. A supply-chain attack against a common Node.js package can be as cheap as a ____________________ and as easy as a modified ____________________, introducing vulnerabilities into tens of thousands of other packages. A supply-chain attack against a motherboard, however (also described here) takes a lot more work, both to implement and then to exploit. However, it is also much more difficult to defend against!

Our goal, then, is not to ____________________ but to ____________________. The less we have to depend on, the better.

# Today

Abstraction and its problems

Trust and TCBs

## Next time:

Software security