ECE560
Computer and Information Security

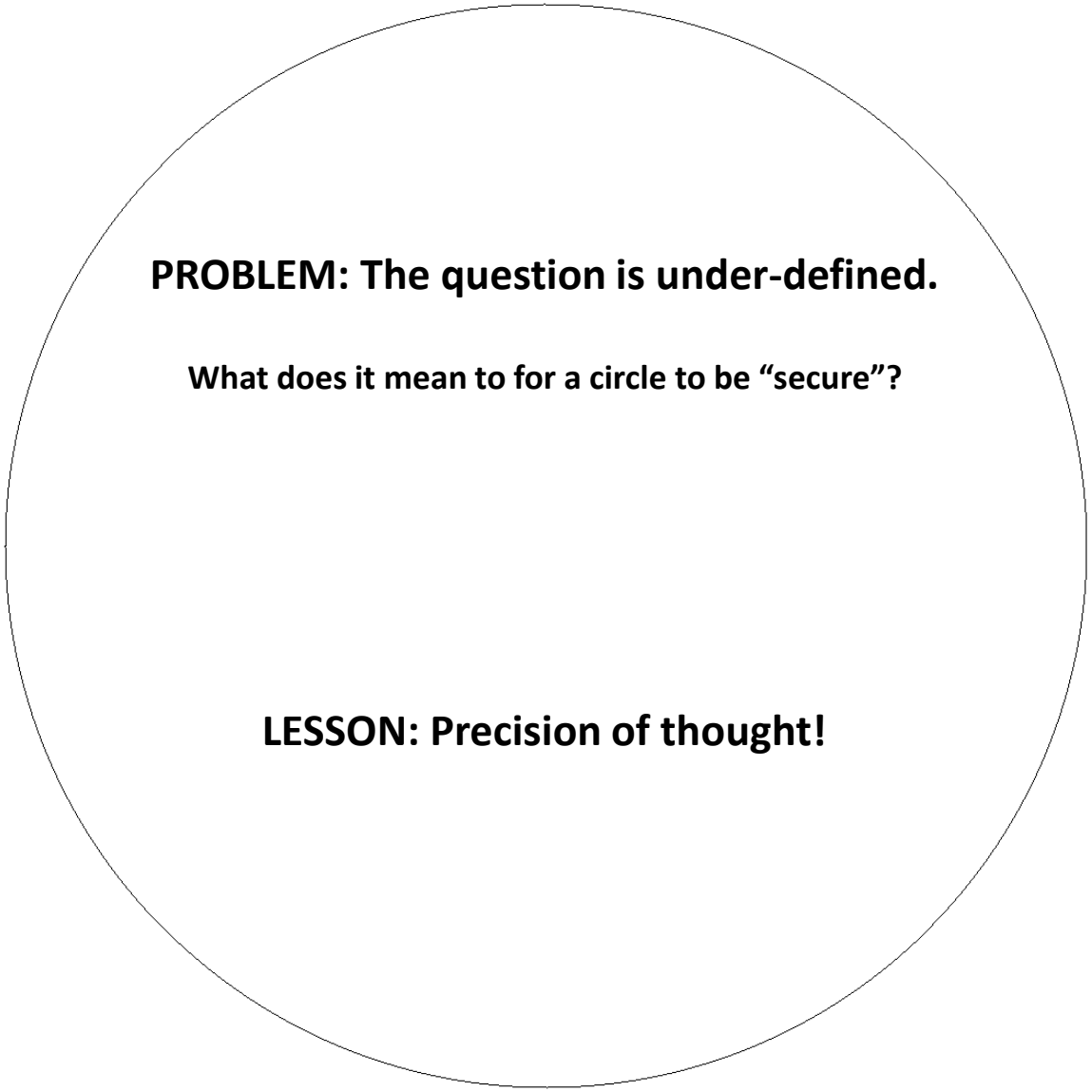
Fall 2020

Computer Security Overview

Tyler Bletsch

Duke University

Is this circle secure?

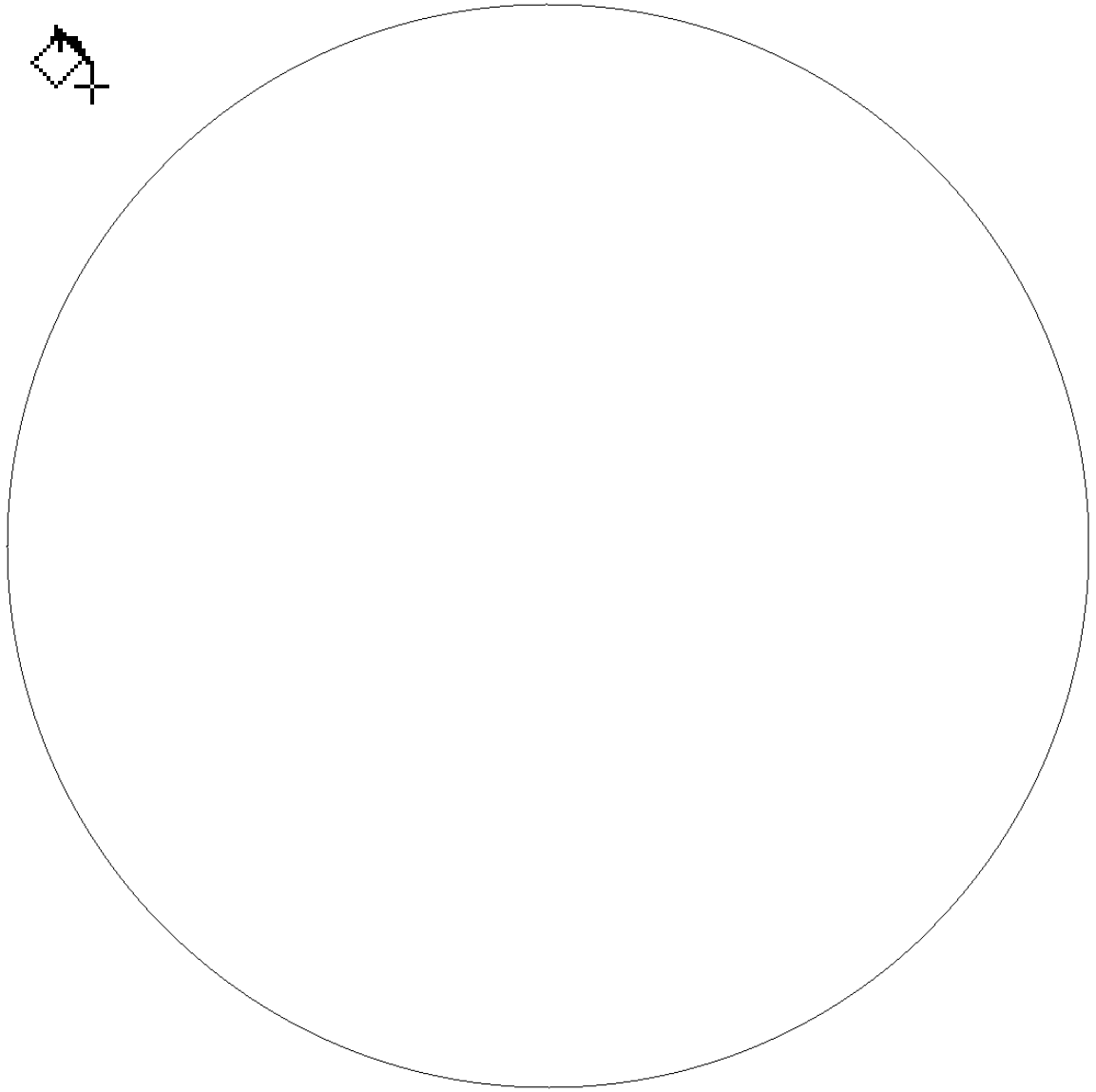


PROBLEM: The question is under-defined.

What does it mean to for a circle to be “secure”?

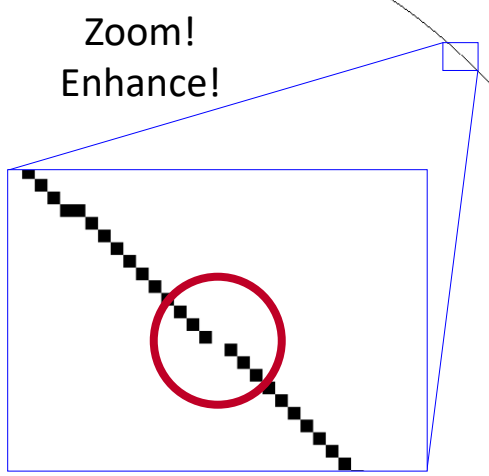
LESSON: Precision of thought!

If I flood-fill outside the circle, will the color penetrate it?



If I flood-fill outside the circle, will the color penetrate it?

Why?



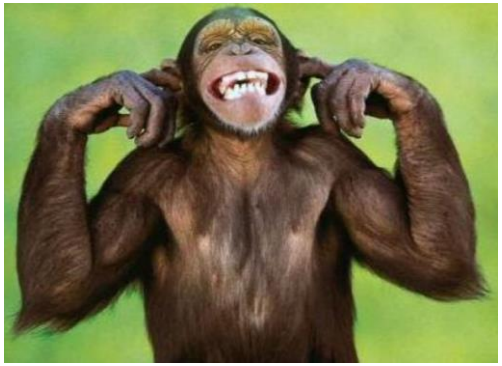
PROBLEM: The defender needs 3000 perfect pixels, but the attacker just needs one flaw.

In computers, you need way more than just 3000 things to be right.

LESSON: Perfect security is usually impossible to prove.

Why that exercise?

- Why did we do this exercise? To put you in the right mindset.
  - We're about to define security and present the fundamental model for reasoning about it.
  - It will seem simple. You will be tempted to ignore it.
    - If you take that mental shortcut, you are inviting ruin.
  - If you want a perfect circle,
    you have to make it **SYSTEMATICALLY AND PRECISELY**
  - ***Security models help us flawed humans avoid missing something!***



We're like 99% this dude. We're not securing anything with our stupid monkey instincts. We need systematic thinking or we're going to make mistakes based on intuition.

What is information security?

From "An Introduction to Information Security" (NIST Special Publication 800-12):

- Information Security:

  The protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to ensure confidentiality, integrity, and availability.

The CIA Triad

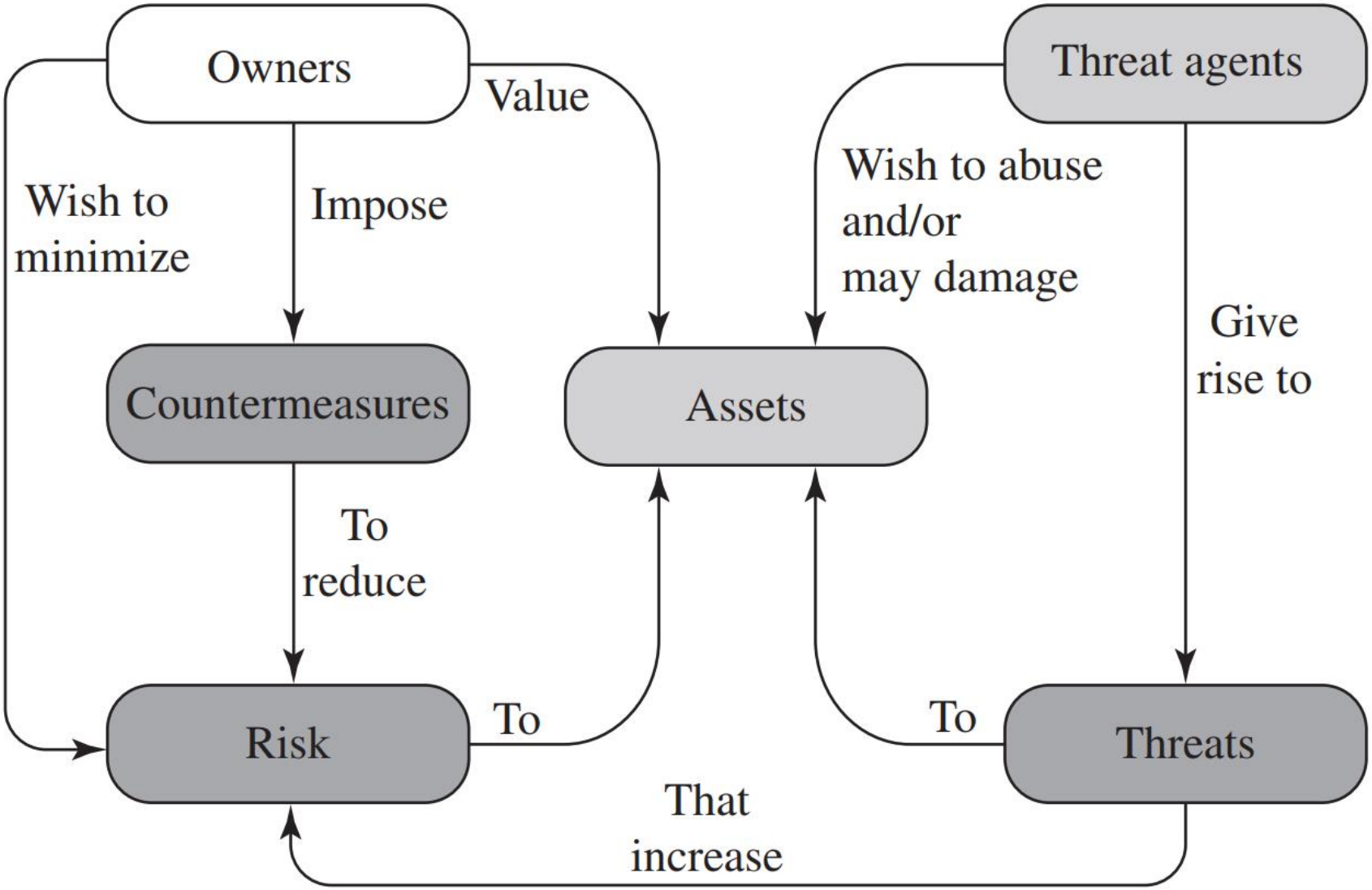
There are like 900 pictures of the CIA triad on google, but this was the ugliest one.

The CIA triad

- **Confidentiality**: Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information.
- **Integrity**: Guarding against improper information modification or destruction and ensuring information non-repudiation[1] and authenticity.
  - Data Integrity – The property that data has not been altered in an unauthorized manner. Data integrity covers data in storage, during processing, and while in transit.
  - System Integrity – The quality that a system has when it performs its intended function in an unimpaired manner, free from unauthorized manipulation of the system, whether intentional or accidental.
- **Availability**: Ensuring timely and reliable access to and use of information.

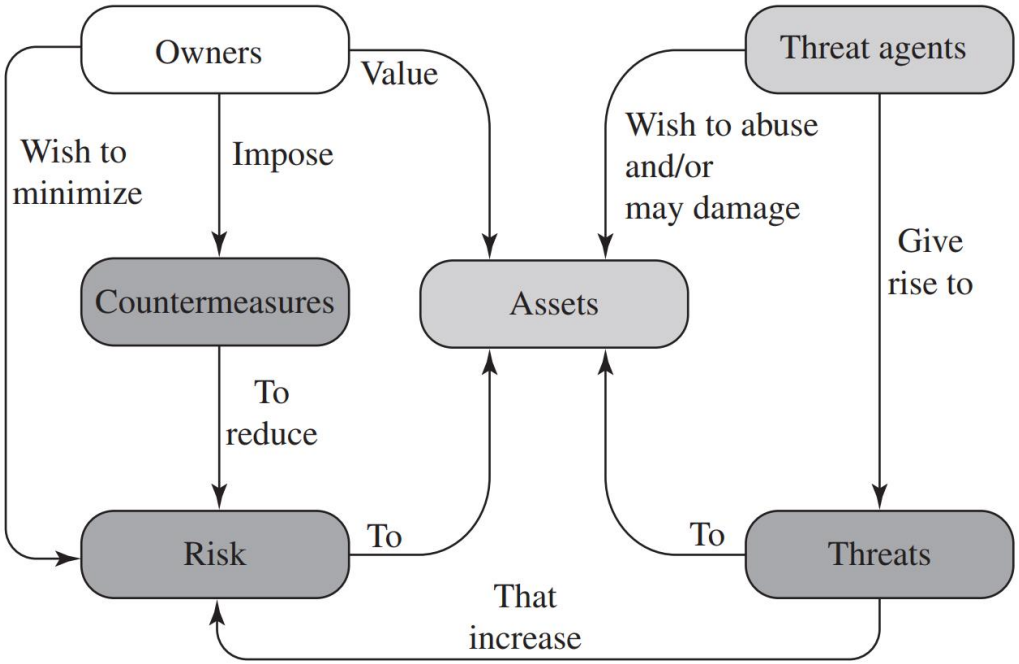
[1] Can positively confirm the source or author of the data.

Computer Security Model



Components of the Computer Security Model

- **Assets**: The valued hardware, software, data, and communications.
- **Threats**: *Specific* attacks against an asset.
- **Countermeasures**: *General* defenses for an asset.
- **Risk**: We don't know the threats, so we summarize our perception of exposure to threats as *risk*.



How do threats work?

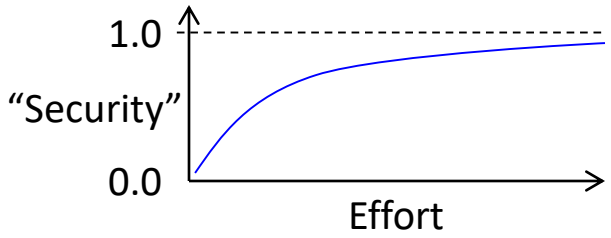
- **Threats** exploit one or more **vulnerabilities** of the asset.
  - Vulnerability may be a design flaw (e.g. a bug or misconfiguration) or a resource constraint (e.g. amount of server resources).
- An **attack** is a threat that is carried out leading to a violation of CIA triad:
  - Information leakage (failure of confidentiality)
  - Doing the wrong thing or giving wrong answer (failure of integrity)
  - Becoming unusable or inaccessible (failure of availability)
- **Countermeasure** deals with a particular class of attack
  - Ideally prevent attack; failing that, at least detect attack and recover.

Thinking about reducing risk

- **Security of a system is boolean**: vulnerable or not vulnerable
- As it is not possible to *prove* the security of a system, **we do not know this boolean's value**
- As such, we apply *countermeasures* to **reduce the probability of attacks succeeding**, given our incomplete knowledge
- This is what we mean by "**reducing risk**"

- This thought process is so common, security professionals may use the verbal shorthand "**this makes the system *more secure***".

- Danger of this shorthand: implies that if you do it enough, you reach "secure". You don't.



“More secure” vs “secure”

“More secure”
(a real concept)

=

“Has countermeasures which, all things being equal, reduce the probability of an exploitable vulnerability being available to attackers, but this probability never reaches zero.”

“Fully secure”
(a fool’s delusion)

- I’m racking up Security Points and if I get enough I win security!
- If I deploy this one thing, I am entirely secure.
- It’s so *simple* we don’t have to think about it!

Classes of threats (1)

RFC4949 defines four broad classes of attack (with sub-types):

1. **Unauthorized disclosure**
   - **Exposure** of sensitive information intentionally (e.g. from insider)
   - **Interception** of info in transit (e.g. network sniffing)
   - **Inference** of info given public data (e.g. an exercise app shows popular exercise locations; this reveals base locations in warzones)
   - **Intrusion** into the system (traditional "hacking" into a server)
2. **Deception**
   - **Masquerade** as someone else (e.g. forging the sender on an email asking for something)
   - **Falsification** of data (e.g. changing your homework grade in Sakai)
   - **Repudiation**: denying you send/received particular data (e.g. "I didn't tweet that, I was ~*hacked*~!")

Classes of threats (2)

RFC4949 defines four broad classes of attack (with sub-types):

3. **Disruption**
   - **Incapacitation** of a system (e.g. denial-of-service attack)
   - **Corruption** of data (e.g. “my username is `";DROP ALL TABLES;--`”)
   - **Obstructing** communications (e.g. wifi jamming)
4. **Usurpation**
   - **Misappropriation** of service (e.g. Captain Crunch’s use of telephone services)
   - **Misuse** of service (e.g. misconfiguring a mail system so it floods someone with email)

Matching assets against the CIA triad

	Availability	Confidentiality	Integrity
Hardware	Equipment stolen/disabled	Physical media stolen	Hardware modified to include tracking or control (e.g. keylogger or keyboard emulator)
Software	OS or program files corrupted, causing loss of service	Proprietary software is stolen	Software is modified to include tracking or malicious control (e.g. malware)
Data	Database or files deleted or corrupted, causing loss of service	Unauthorized reading of user data	Files are modified by malicious actor
Communications	Messages blocked or communication line damaged or shut down	Messages intercepted and read or traffic pattern is analyzed	Messages are modified, duplicated, fabricated, or otherwise molested in transit.

FIPS 200 requirements (1)

FIPS 200 (government document) defines high level security requirements

- **Access control**: Limit who gets in and what they can do
- **Awareness and training**: Prevent uninformed users aiding attack
- **Auditing and accountability**: Track who's doing what
- **Certification and assessment**: Periodically review security posture
- **Config management**: Track how things are configured, note changes
- **Contingency management**: Have plans for emergencies
- **Identification/authorization**: Check user identities
- **Incident response**: Plan how to respond during/after a breach
- **Maintenance**: Actively maintain systems (no deploy & forget!)

Note: This is hyper-summarized, of course

FIPS 200 requirements (2)

FIPS 200 (government document) defines high level security requirements

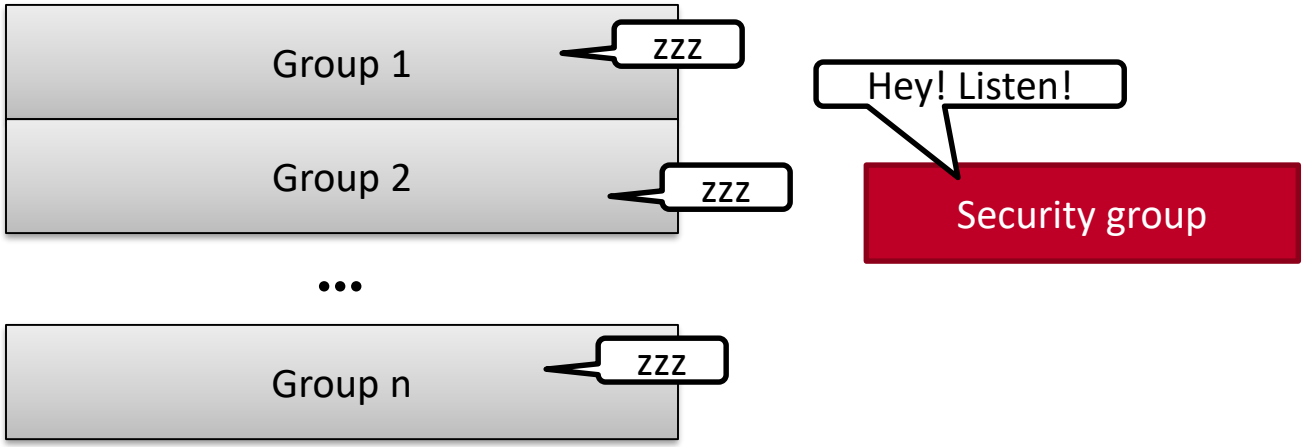
- **Media protection**: Keep storage safe (even when trashing it)
- **Physical/environmental protection**: Doors, walls, cameras, etc.
- **Planning**: Every action is planned then executed, no 'cowboy IT'
- **Personnel security**: Vet the people working there
- **Risk assessment**: Analyze risk and invest proportionally
- **System and services acquisition**: Source goods/services wisely
- **System and communication protection**: Good software engineering
- **System and information integrity**: Malware countermeasures

Q: Are these **technical factors** or **human factors**?

Human and technology factors are *interwoven*

- Good model of security: “a thread through everything”

- Bad model of security: “a separate silo”



Design principles for security in software (1)

From *National Centers of Academic Excellence in Information Assurance/Cyber Defense* from U.S. government

- **Economy of mechanism**: Each feature is as small and simple as possible. This makes it easy to reason about and test, and is likely to have fewer exploitable flaws.
- **Fail-safe defaults**: In the absence of a explicit user choice, the configuration should default secure. For example, a daemon that listens to local connections only unless explicitly set to remote access.
- **Complete mediation**: Every access is checked by system; access cannot be "cached" or left up to the client. In other words, take the concept of time out of the equation when thinking about security – all accesses are assessed on the most current configuration.
- **Open design**: Don't keep your design secret – an inspected design is more secure than one you *hope* is secure. Goes against human instinct ("don't let them see our stuff, they might find a problem!").

Design principles for security in software (2)

From *National Centers of Academic Excellence in Information Assurance/Cyber Defense* from U.S. government

- **Separation of privilege**: Define fine-grained privileges in your system (as opposed to one big admin privilege) and separate software so that common functions are done with a lesser privilege level than more sensitive functions.
- **Least privilege**: Give only the specific access a user/component needs to do its job.
- **Least common mechanism**: Minimize sharing of capabilities among users, analogous to "separation of powers" in government.
- **Psychological acceptability**: Don't interfere with human's workflow to such an extent that they break security to get their jobs done. For example, changing a 20-character password every week just makes everyone choose simple incrementing passwords or use post-it notes.

Design principles for security in software (3)

From *National Centers of Academic Excellence in Information Assurance/Cyber Defense* from U.S. government

- **Isolation of systems**: Make low-security public systems separate from high-security ones.
- **Isolation of users**: Users should have separate files, processes, etc. Enforced by modern operating systems.
- **Isolation of security functions**: Security tools and facilities should be separated from production functions where possible.
- **Encapsulation**: Provide software interfaces that allow access to data only through prescribed routes; disallow direct access to underlying data access or objects.
- **Modularity**: Use common software modules for security functions (e.g. cryptography); reduces odds of a “one-off” module’s flaw. Apply modularity generally also so that updates can be done with low risk.

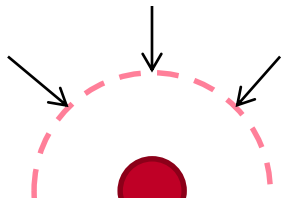
Design principles for security in software (4)

From *National Centers of Academic Excellence in Information Assurance/Cyber Defense* from U.S. government

- **Layering**: Apply multiple overlapping security techniques. Avoid a condition where a single breach compromises everything (such as the flawed concept of the “trusted internal network”).
- **Least astonishment**: Programs should not surprise the user. For example, many UNIX programs use the ‘-h’ flag to mean “help”. You should not write a program where ‘-h’ means “hurry up and delete everything”.

Attack surface

- For a system already deployed, you may want to assess risk
- One thing to ask: how many ways could an attacker interact with it?
- This is the **attack surface**.
  - Includes the software itself, the network, and humans.
  - Examples of attack surfaces in desktop operating systems:
    - Big attack surface: **Windows 95**, when it comes online, listens for connections on several port numbers with various large and complex services.
    - Smaller attack surface: **Windows 10**, when it comes online, listens on a few ports, and has a firewall that blocks most connections (but the firewall has exceptions by default that still allow some fairly complex services to listen).
    - Even smaller attack surface: **Ubuntu Linux 18.04**, when it comes online, listens on no ports whatsoever.
- **Good practice: find ways to reduce attack surface!**



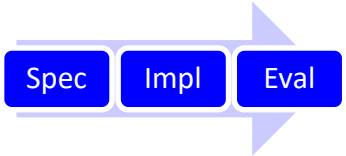
Security strategy

1. **Specification/policy**: What is your goal? Consider tradeoffs against ease of use and cost.

2. **Implementation**: Identify mechanisms of prevention, detection, response, and recovery.

3. **Evaluation**: Don't assume it worked; prove it.

- The above seems simple, but ***I have seen one of these steps skipped SO MANY TIMES.***
  - I've seen people forget #1 (deploy and evaluate tools without regard for their needs)
  - I've seen people forget #2 (decide on goals, not fund the implementation, then get mad when they're not met)
  - I've seen people forget #3 (set up fire-and-forget security solutions that quietly die soon after)



Threat modeling

- When designing a defense, you have to know the goal
- Define:
  - **Asset(s)** at risk
  - Type of **vulnerability** you assume exists and are protecting against
  - **Attacker's capabilities/knowledge**
- Only then can you say how your defense prevents the attack from succeeding despite the vulnerability (or detects it, response to it, or recovers from it).

Threat modeling example: HTTPS

HTTPS: Encrypted form of HTTP for secure web traffic

Threat model:

- **Asset(s)**: Private user communications, including credentials
- **Vulnerability**: Packets may be intercepted in transit (e.g. on open wifi)
- **Attacker's capabilities/knowledge**: Knows when/how to intercept packets for a specific user or for the site as a whole

The defense:

- **Our solution**: we negotiate a key in open communication known only to user and server; all content is encrypted with this key.
- **How it solves it**: Even with the full traffic, attacker cannot deduce key and therefore cannot decrypt communications. However, they do know that communication happened and roughly how much...

Why threat model?

- Threat models help us move from nebulous world of “more secure” to a specific guarantee
- MOST IMPORTANT: Promotes **systematic thinking** about when a defense can and cannot do
- Lets us compare techniques in terms of cost/benefit tradeoffs
- Understand what attacks are still on the table

Conclusion

- Perfect security is impossible
- Constant struggle to ensure *everything* is correct;
  attacker just has to find a *single* flaw
- We do our best using **systematic thinking** guided by models, e.g.:
  - The CIA triad
  - The information security model (asset/vulnerability/threat/attack)
  - Security strategy model (specify/implement/evaluate)
  - Attack surface modeling
  - Threat modeling (asset/vulnerability/attacker)
- Reduce likelihood of missing something with design principles, e.g.:
  - FIPS 200 security requirements (human and technical factors alike!)
  - Design principles for security in software design