



# Lecture 2: Measuring Security

*presented by*

**Li Yi**

*Assistant Professor*  
SCSE

N4-02b-64

[yi\\_li@ntu.edu.sg](mailto:yi_li@ntu.edu.sg)

# Security Metrics

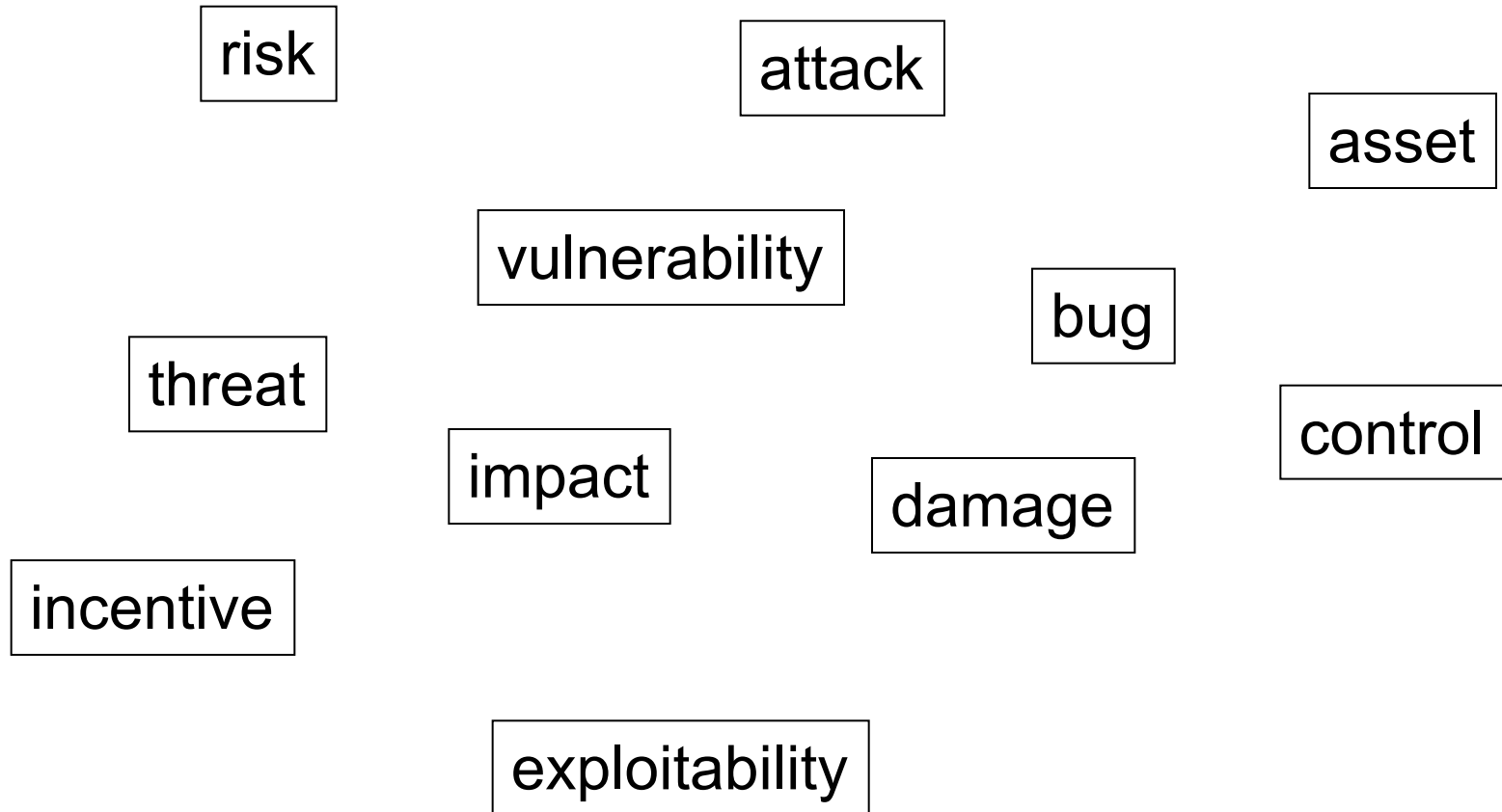
“To make progress, you need security metrics, no matter what they measure!” [Butler Lampson]

- You need to measure the progress you are making in securing your system
- You might want to convince your manager that your efforts had a measurable impact
- Being able to compare the security of different systems would be nice
- To meet requirements/regulations and maintain compliance (e.g., ISO27001 and ISO27005)
- Not very satisfying from an academic point of view, but important in practice

# Agenda

- Risk analysis
- Risk analysis for software development
  - STRIDE
  - Threat trees
  - DREAD
  - CVSS – scoring vulnerabilities
- Attack surfaces
- Summary

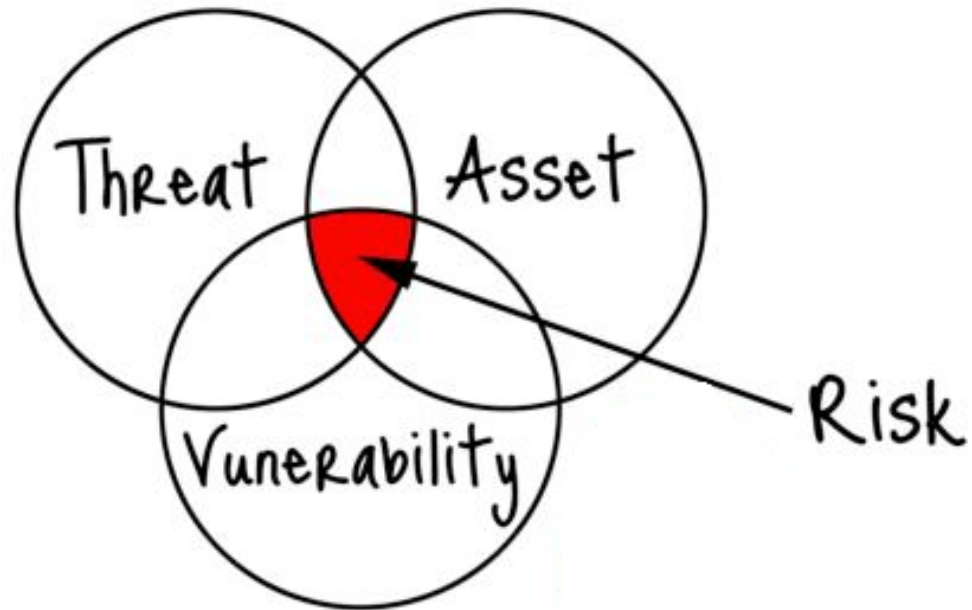
# Relationships?



# Risks, Threats, Vulnerabilities

- **Vulnerabilities** are weaknesses of a system that could be accidentally or intentionally **exploited** to damage assets
- **Threat** is a potential event that will have unwanted consequences if it becomes a reality
- **Attacks** are actions by adversaries who try to exploit vulnerabilities to **damage** assets
- **Risk** is the possibility that some incident or attack can cause damage to your enterprise
- To assess the risk posed by an attack, evaluate the amount of damage being done (**impact**) and the **likelihood** for the attack to occur and succeed

# Risk Analysis



Risk analysis enables you to **identify assets, vulnerabilities, and threats** in order to make informed decisions about which **control (mitigation)** to use

Image: <http://www.safezoneafrica.com/security-risk-assessment/>

# Risk Analysis

- **Risk analysis:** assessing the consequences of **uncertain** events
  - There are known unknowns and unknown unknowns ...
  - Many related concepts are relevant for risk analysis, but we cannot offer universally agreed standard definitions of those concepts
  - It is important to approach risk analysis **in an organized fashion**; beyond being organized, no deep science is being asked for

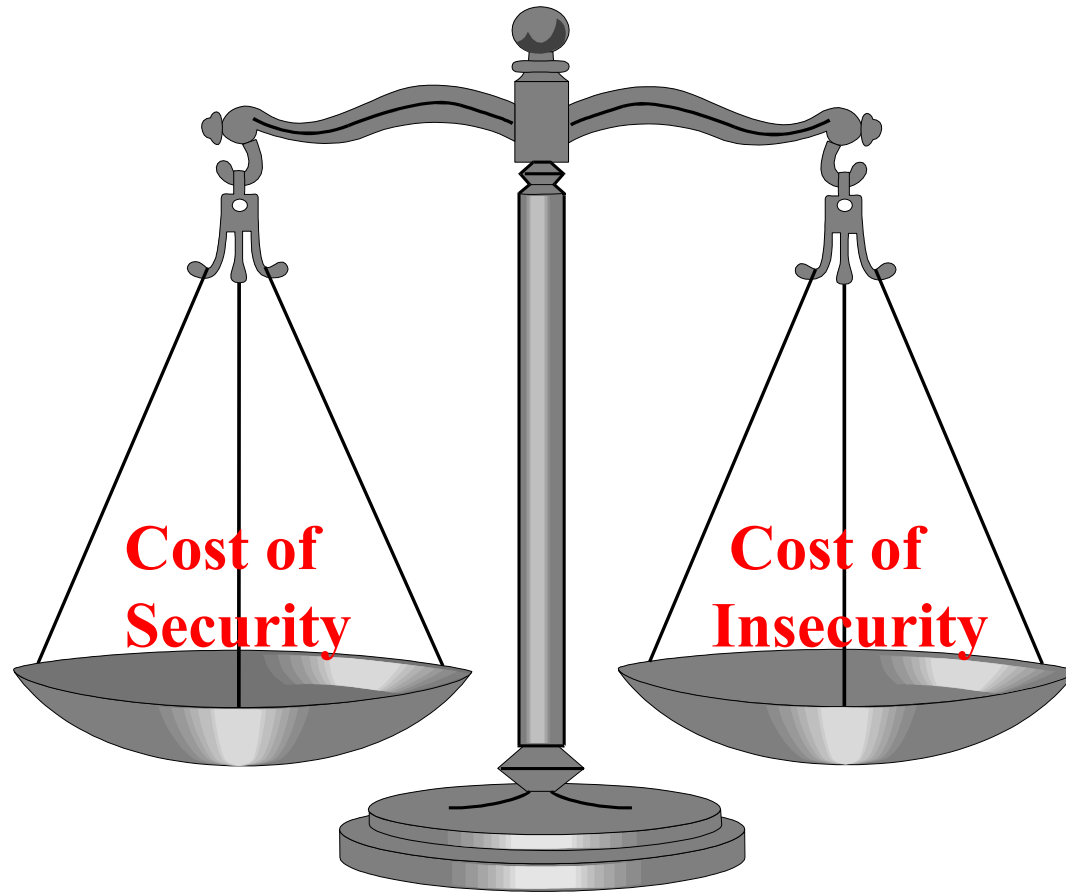
# Risk Analysis



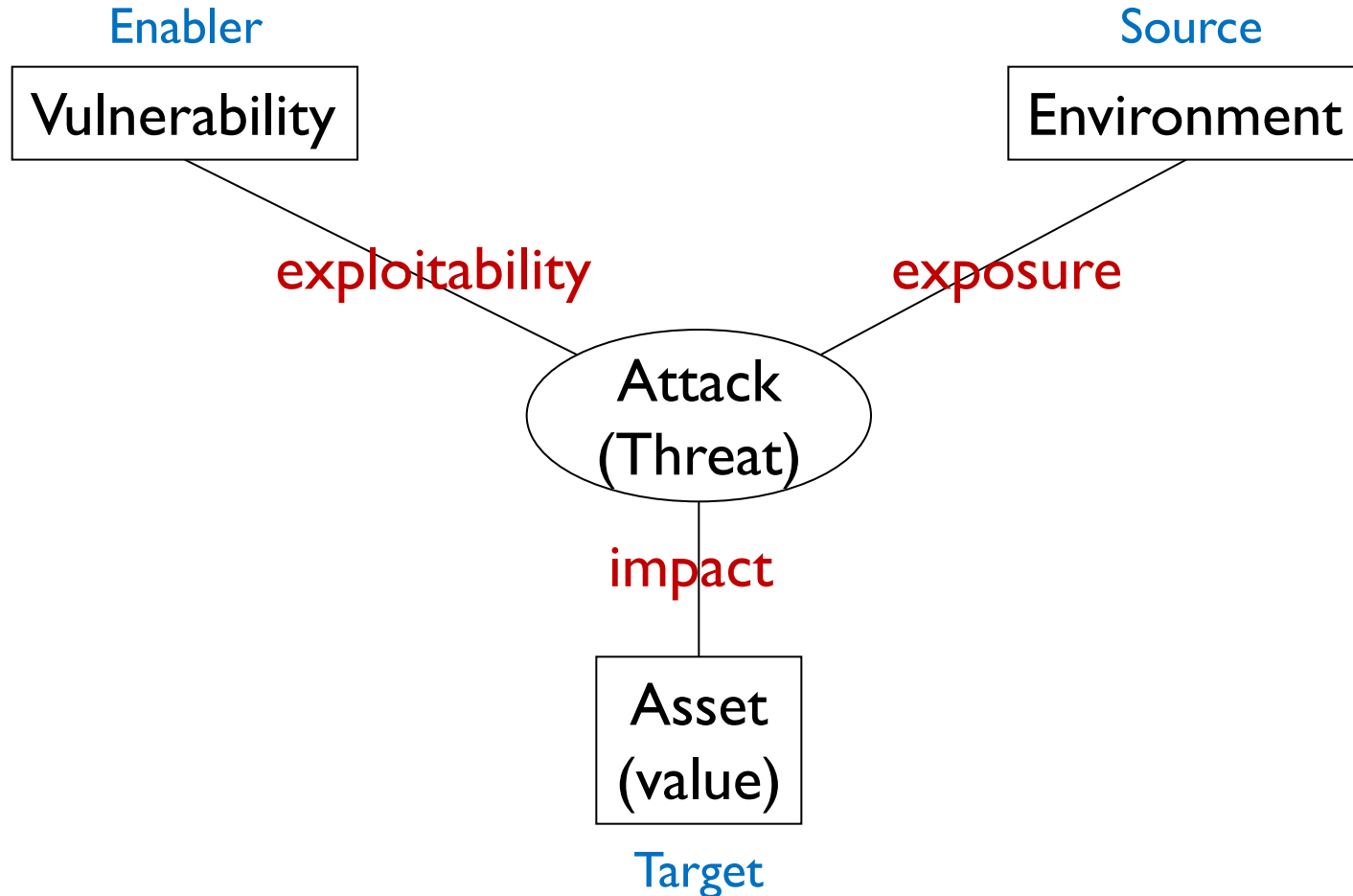
<https://www.cartoonstock.com/directory/r/risk.asp>



# Risk Analysis – Balancing the Costs



# Factors in Risk Analysis



# Risk Analysis Cont'd

- Risk analysis can be performed
  - When developing new software components
  - For the IT infrastructure of an organization
  - For all information assets of an organization

# Risk Analysis in Software Development




You cannot build a secure system until you understand your threats

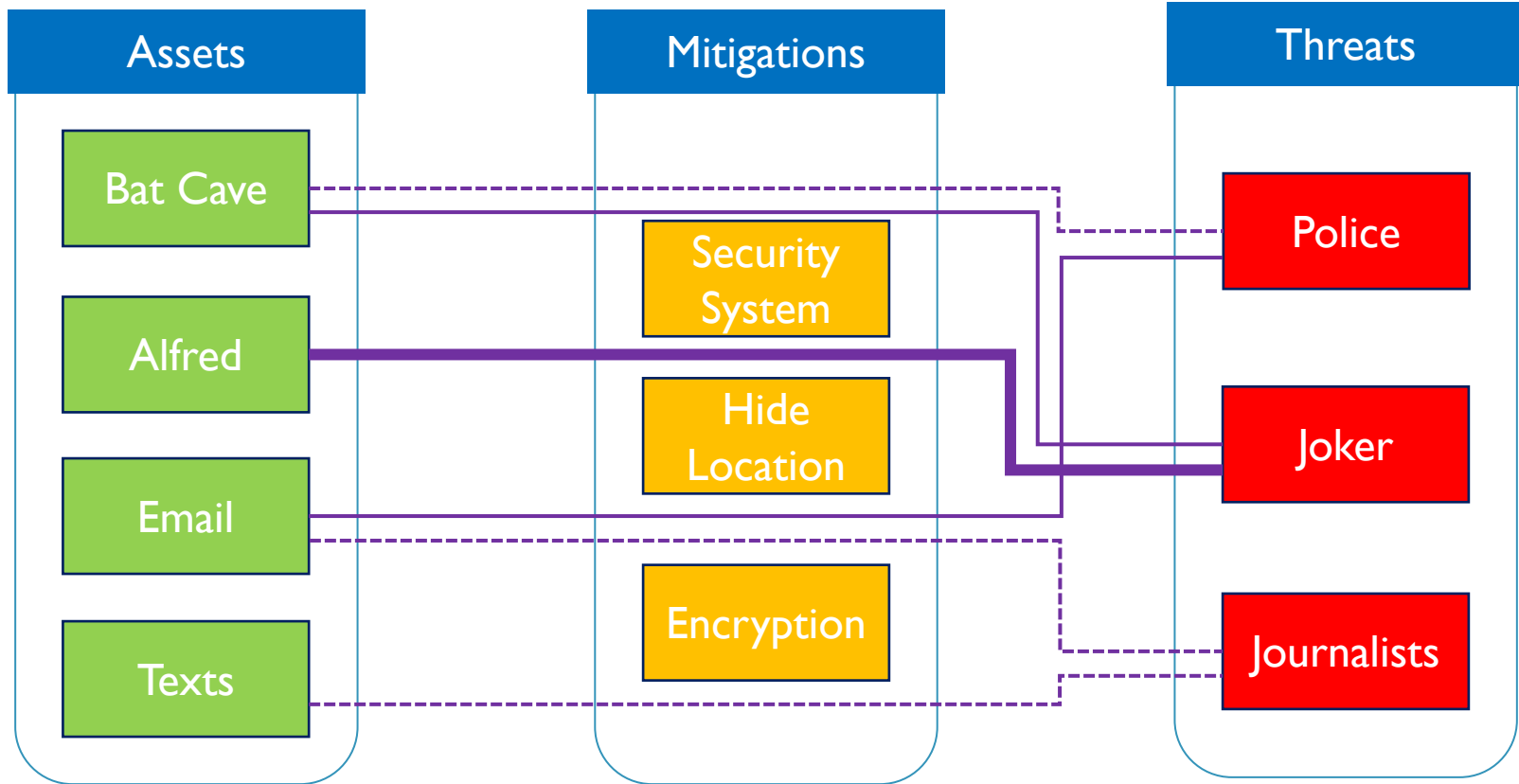
# Threat Modelling

We will now examine how security could be considered in the design phase **in a structured manner**

- To structure security analysis, create a **threat model**
  - Assumptions about a hypothetical adversary
- Threat modelling
  - What are the threats?
  - Which threats need mitigating?
  - How to mitigate those threats?

# Bruce Wayne (Batman)'s Threat Model

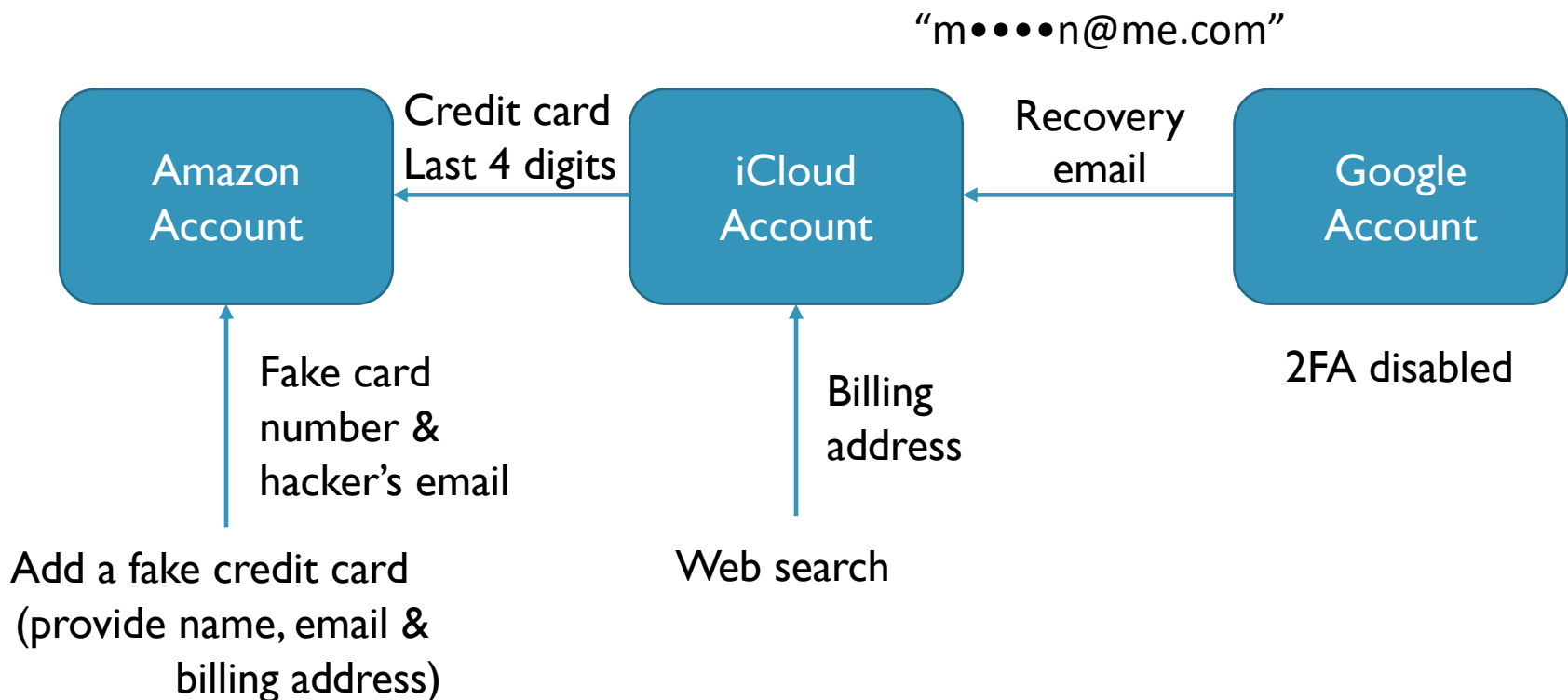
Vectors	
	Low Risk
	Med Risk
	High Risk



<https://arstechnica.com/information-technology/2017/07/how-i-learned-to-stop-worrying-mostly-and-love-my-threat-model/>

# Threat Modelling

- Incomplete/inaccurate threat model lead to security flaws
- The story of “epic hacking”: <https://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/>



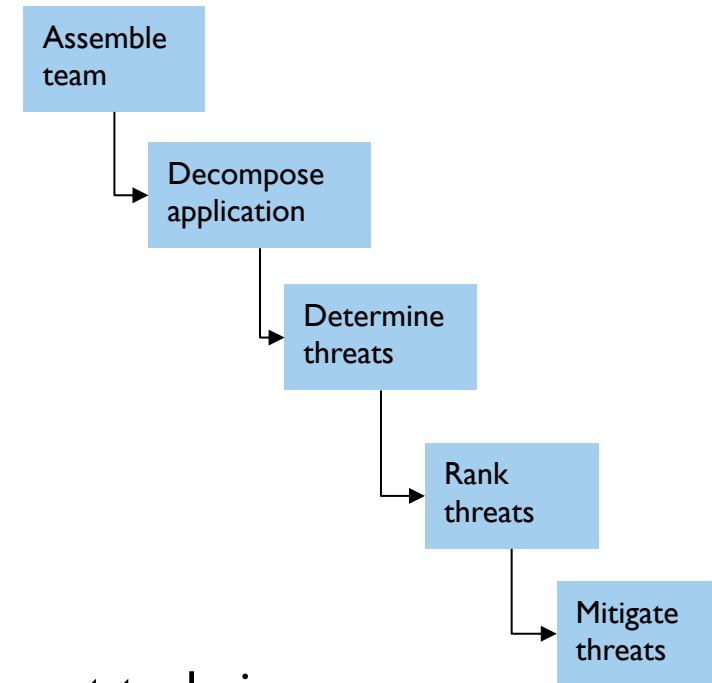
# Threat Modelling

- Gets you to think about the security of your application in a relatively formal way
- Helps understand where the product is most vulnerable and chooses appropriate techniques to mitigate threats, which leads to more secure systems
- Helps find bugs, in particular those design bugs that are not likely to be found when looking at individual components
- Threat models can help new team members understand the application in detail
- Threat models should be read by other product teams that build on your product. This is important information when building a system from a number of components
- Threat models are useful for testers, too



# Threat Modelling Process

1. Assemble threat modelling team
2. Decompose application
3. Determine threats
4. Rank threats
5. Mitigate threats
  - Decide how to respond to threats
  - Decide on techniques to mitigate threats
  - Choose appropriate technologies to implement techniques



[Chapter 4 of Howard & LeBlanc: “Writing Secure Code”]

# Assemble Threat Modeling Team

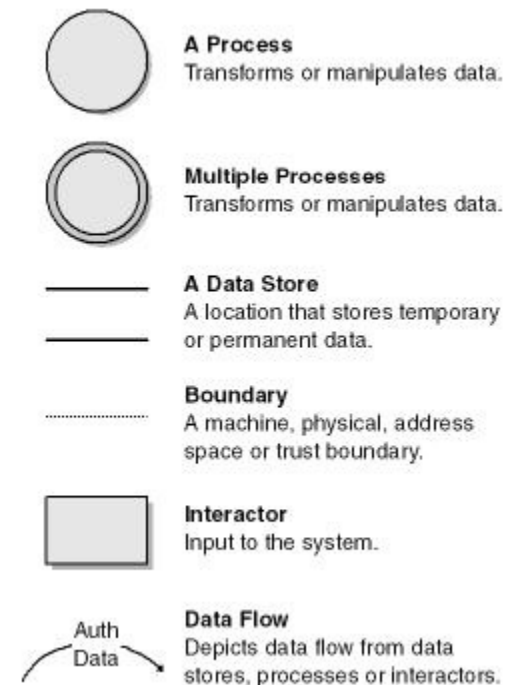
- Gather together people from the product group and have regular threat modelling meetings
- Have most security-savvy person lead the team – person able to look at a given application or design and work out how an attacker could compromise the system, knowledgeable about failures in the past
- Have one member involved at the meetings – design, coding, testing, documentation
- Have other group review the result and invite marketing or sales persons to the meetings (so that they form part of the process and able to explain to the clients about what you are doing)
- Do not try to fix problems during the threats-modeling meetings

# Formally Decompose Application

- Identify key **components**
- Identify relevant security **boundaries**
- Identify **data flow** between components
  - Howard & LeBlanc: **Data flow diagrams** (DFDs) found to be more useful than UML activity diagrams (control flows)
- Repeat this process
  - Start from high level view
  - Decompose components into subcomponents
  - Decompose subcomponents ...

# DFDs for Decomposing the System

- Principle of DFDs: a system can be decomposed into subsystems, and subsystems can be decomposed into even lower-level subsystems
- First, determine the boundaries or scope of the system and **identify the boundaries** between trusted and untrusted components, by using a high level **context diagram**

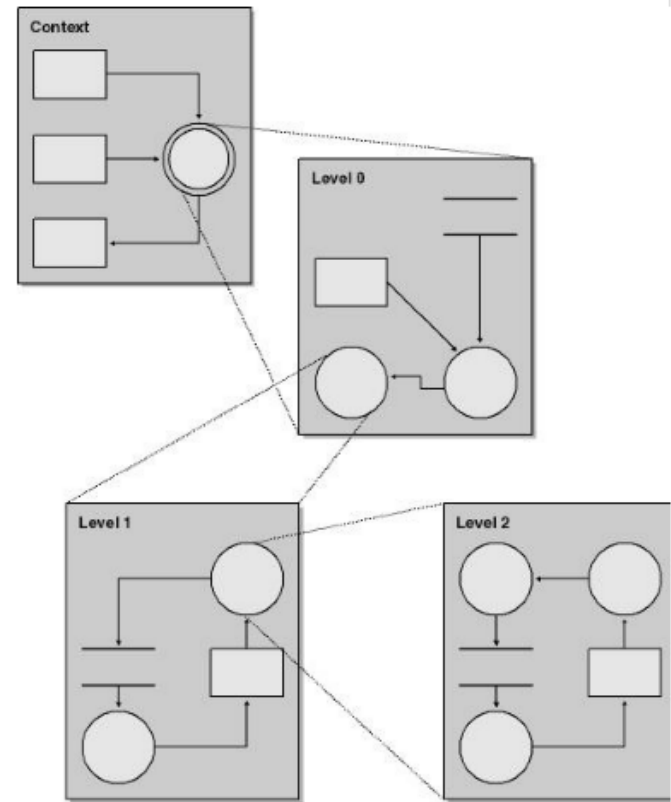


Key DFD symbols

[Chapter 4 of Howard & LeBlanc: “Writing Secure Code”]

# DFDs for Decomposing the System

- **Context diagram** usually contains only one process and no data store, modeling the **users** or **external entities** interacting with the system.
- Then, the system is decomposed into levels by using level-0, level-1, and level-2 diagrams.
- This process is just like a typical software design process; not something new.

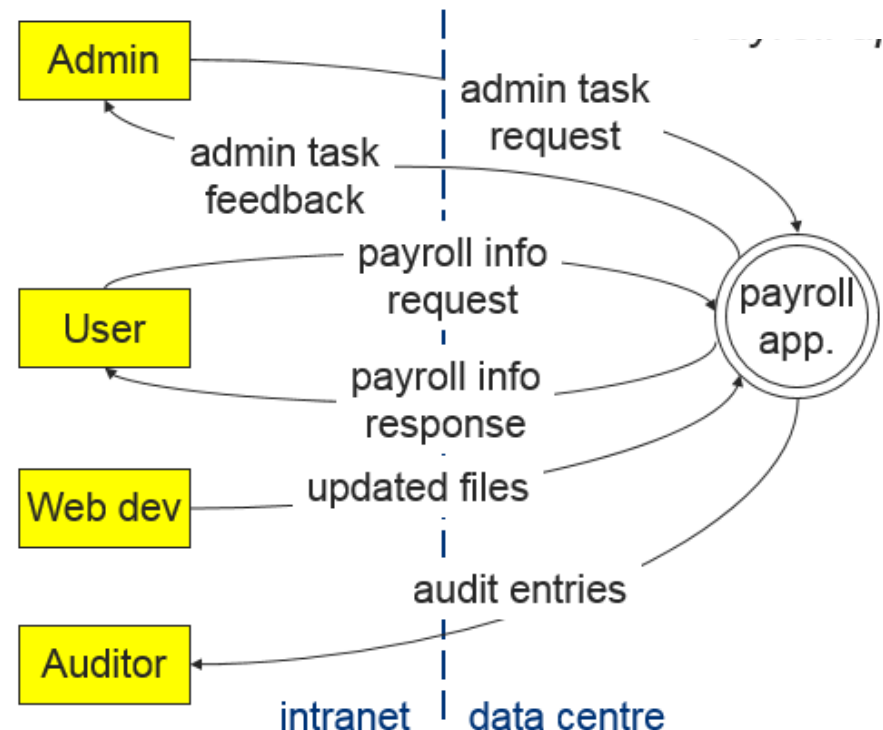


General concept of DFDs—drilling down from a context diagram to lower level data flow

# Example: context diagram for the sample payroll application

Questions to think about:

- To what events or **requests** must the system respond?
- What **responses** will the process generate (service provided)?
- Who is the **recipient** of each response?
- What are the **data sources** related to requests and responses?
  - Persistent (file, database) and non-persistent data (session)



# Example: Level-I DF

- Following **naming conventions**
- For threat modeling, two, three or four levels are generally enough to understand the composition of the application
- **Just go deep enough**, not to spend more time than necessary (analysis paralysis)
- Remember our goal is to analyze threats...

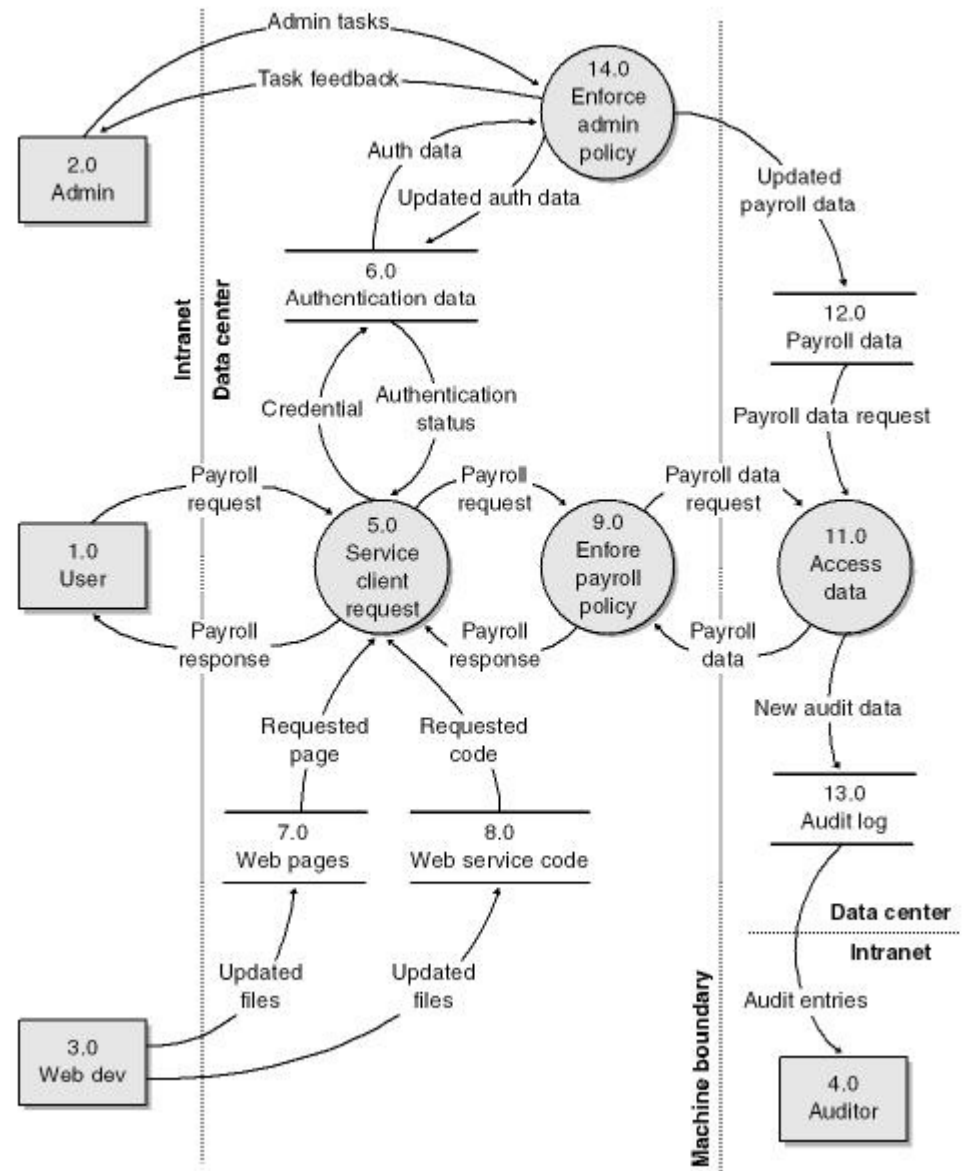


Image courtesy of Chapter 4 of Howard & LeBlanc: "Writing Secure Code"

# Comments

- Understanding the **components** of a software system and the **interfaces** between them can help to identify problem areas
- **Documenting the assumptions** made during risk and threat analysis can tell other development teams which problems you had been aware of
- In particular, this would alert them to issues they have identified but you had not dealt with



# STRIDE

Determine and categorize threats to the system

# Threat targets

- In the next step, examine the DFDs for potential threats to the application
- Again, it helps to organize threats analysis
- Take the identified components or assets (e.g. data store that stores authentication data) from the decomposition process and use them as the *threat targets*

# Formulating Threats

- Look into each **threat target** and ask:
  - Can a non-authorized user **view** the confidential network data?
  - Can an untrusted user **modify** the patient record data in the database?
  - Could someone **deny** valid users service from the application?
  - Could someone take advantage of the feature or component to **raise their privileges** to that of an administrator?

# STRIDE: one proposal for categorizing threats

- **S**poofing Identity
- **T**ampering with data
- **R**epudiation
- **I**nformation disclosure
- **D**enial of service
- **E**levation of privilege

➤ Other threat analysis techniques include: OCTAVE from Carnegie Mellon University (<http://www.cert.org/octave>)

# Spoofing Identity

- Attacker poses as another entity (user, server)
- Examples:
  - HTTP authentication (basic and digest authentication can be easily spoofed)
  - Compromised passwords, PINs
  - DNS spoofing
  - Spoofed email addresses, ...
- Also known as **impersonation** and **masquerading**
  - **Identity theft** is a related issue

# Tampering with Data

- Unauthorized or malicious modification of data
- Examples:
  - Insufficiently protected (e.g. world-writeable) files and database entries
  - Unprotected transmission of data over a network
- Also known as (data) **integrity**

# Repudiation

- User (wrongly) denies having performed an action, but there is insufficient evidence to prove the contrary
- **Non-repudiation**: providing ‘irrefutable’ evidence to counter repudiation threats
- The interpretation of ‘irrefutable’ and ‘prove’ depends on the application
  - A proof in court is not the same as a mathematical proof
- Also known as **accountability**, related to **attribution**

# Information Disclosure

- Information revealed to unauthorised entity
- Examples:
  - Read access to insufficiently protected (e.g. world readable) files or database entries
  - World-readable files of hashed passwords?
  - Unprotected transmission of data over a network
- We might also want to hide the existence of certain files or events, not only their content
- Also known as confidentiality



# Denial of Service

- Deny service to an authorised user
- Examples:
  - Crashing a system (ping flooding)
  - Distributed Denial of Service attacks (DDoS)
  - All DDoS = DoS but not all DoS = DDoS
- Can be anonymous
- Can rarely be stopped completely but can be made more expensive for an attacker
- Also known as **availability**

# Elevation of Privilege

- User gains more privileges than entitled, thereby has sufficient access to compromise or destroy the entire system
- Examples:
  - Root attacks
  - Trojan horses: run with the victim's privileges
  - Vulnerable Linux SUID programs: <https://www.hackingarticles.in/linux-privilege-escalation-using-suid-binaries/>
  - XSS attacks (Lecture 9)
- Related to various spoofing attacks

# Comments

- STRIDE categorizes threats by their **effects**
- We could also classify threats by their **causes** (e.g., absence of array bound checking)
- The threats listed are **interrelated**
  - Information disclosure could lead to spoofing threats; a spoofing attack may lead to elevation of privileges, etc.
- Lists are rarely complete and you might be able to come up with threats that do not fit in
- A good acronym can help raising the level of awareness on various security issues
- STRIDE is popular also outside software security

# Threat Trees

Visualizing how an asset can be attacked

# Threat Trees

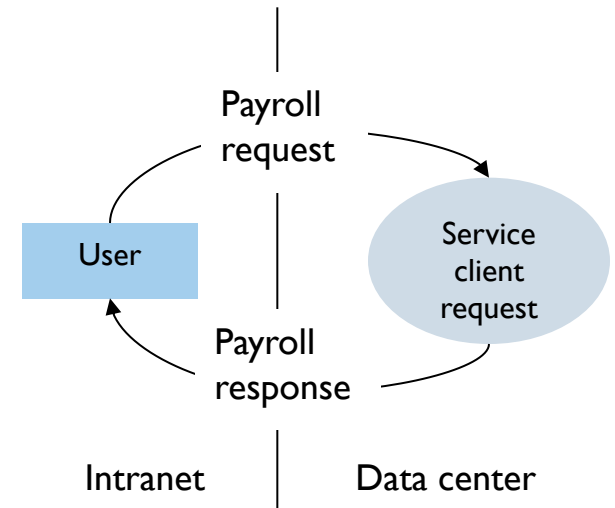
- To further organize threat analysis, construct **threat trees** (**attack trees**)
- Threat tree describes the decision-making process an attacker would go through to compromise an asset
- When the decomposition process gives you an inventory of application components (assets), you start **identifying threats** (**using STRIDE**) to each of them

# Threat Trees Cont'd.

- Once you identify a potential threat you are concerned about, you then determine **how that threat could manifest itself** by using threat trees
- For each threat, “think like an attacker”; list sub-goals an attacker has to reach to realize the threat
- Repeat this process for each sub-goal, until you reach a level of detail matching your security mechanisms
  - Again, you have to know when to stop
- **Attack-defence trees** capture countermeasures as well

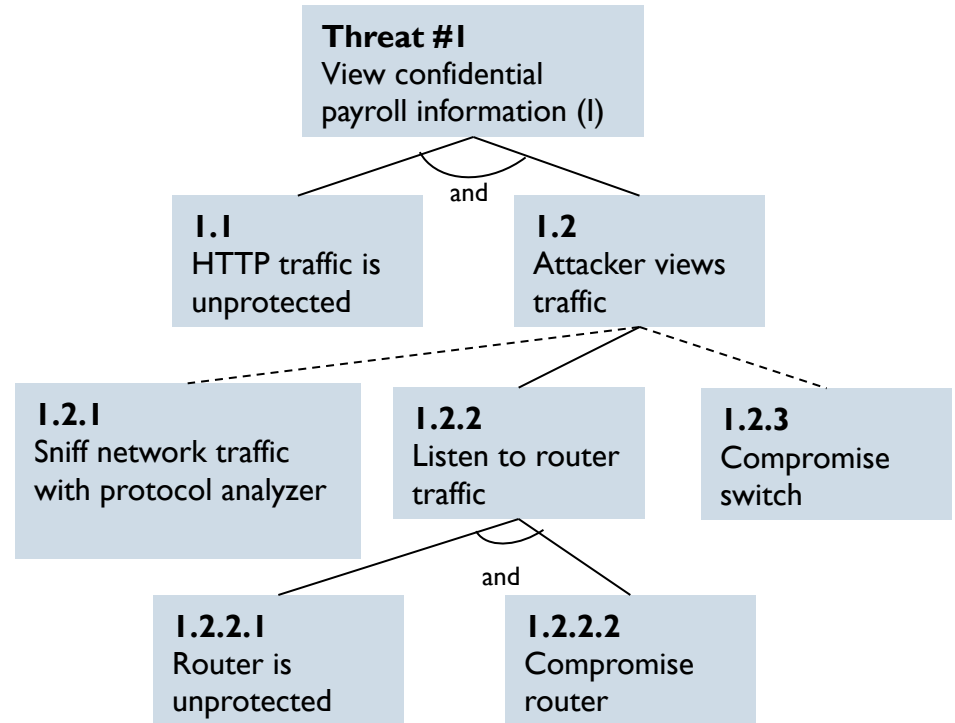
# Example: Threat Tree

- The payroll data that flows between the user (an employee) and the computer systems inside the data centre
- It's sensitive data, confidential between the company and its employees — you don't want a malicious user looking at someone else's payroll information
- This is an example of an **information disclosure threat** from STRIDE



# Example: Threat Tree

- Show a number of ways an attacker can view the data
- The top box is the ultimate threat; the boxes below are the **steps involved** to make the threat an attack
- Record the type of threat based on STRIDE or any applicable model





# An Outline View

I.0 View confidential payroll data on the wire

I.1 HTTP traffic is unprotected (AND)

I.2 Attacker views traffic

I.2.1 Sniff network traffic with protocol analyzer

I.2.2 Listen to router traffic

I.2.2.1 Router is unpatched (AND)

I.2.2.2 Compromise router

I.2.3 Compromise switch

Makes large threat trees more readable

# Example: Fine-Tuning Threat Tree

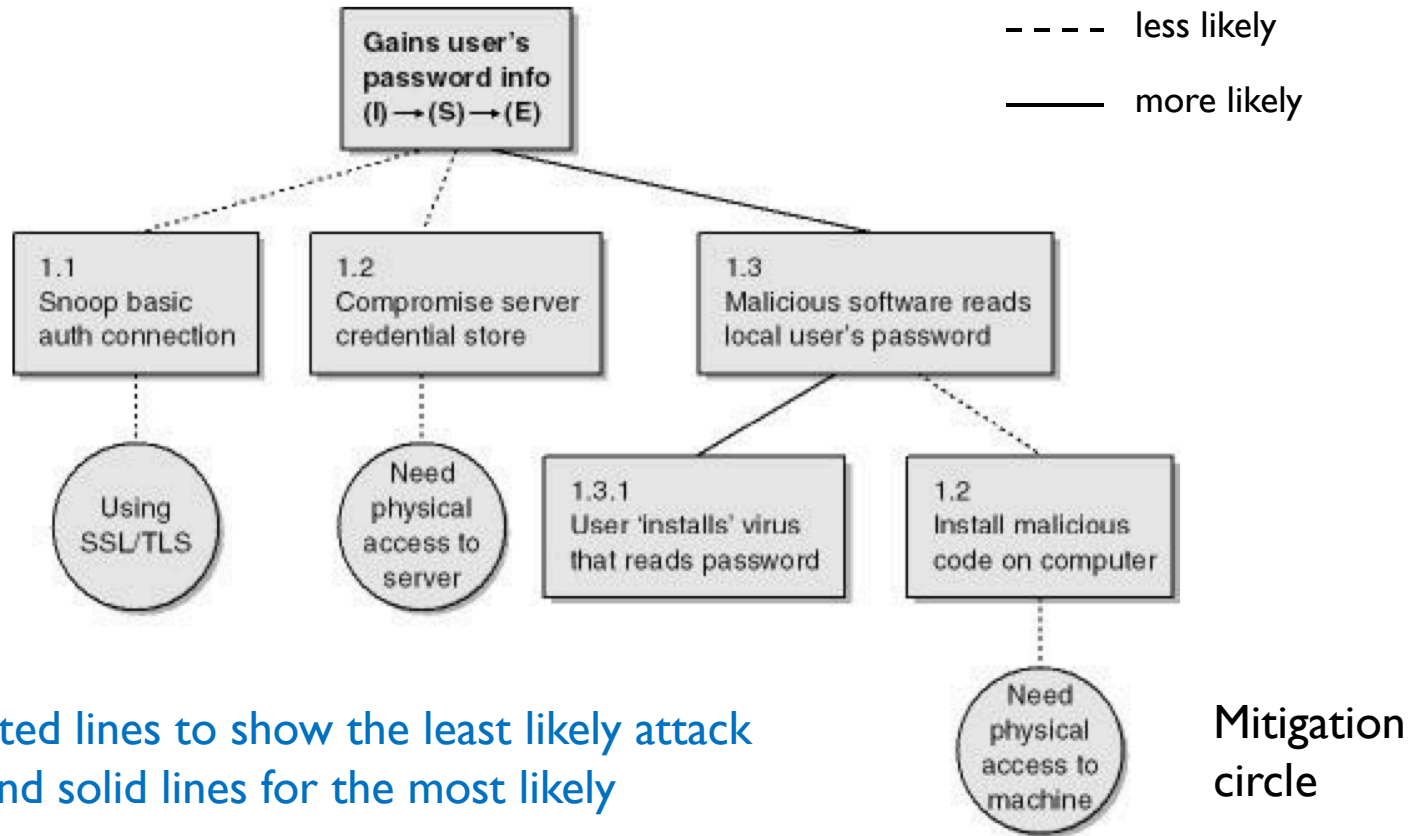


Image courtesy from Chapter 4 of Howard & LeBlanc: "Writing Secure Code"

# DREAD

Ranking the threats by decreasing risk

# Ranking Threats

- Once the threats have been identified, you have to decide what to do about them
- It makes sense to deal with **the greatest risks first**
- Risks are a measure of the damage an attack may cause and the likelihood for it to happen
- Several methodologies for calculating risks from:
  - The value of assets
  - The ease of exploiting a vulnerability
  - The expected likelihood of the threat
- The method you use is not important so long as you are realistic and consistent
- Simple one: ***Risk = DamagePotential \* Likelihood***

# DREAD

- In a risk analysis for software, it may be difficult to properly value assets
- First example: **DREAD** methodology (once used at Microsoft)
- Risks are rated according to:
  - **D**amage potential
  - **R**eproducibility
  - **E**xploitability
  - **A**ffected Users
  - **D**iscoverability

# DREAD

- **D**amage Potential

- How great can the damage be?
- Elevation of privilege is usually very serious (10/10)
- Value of data affected? (medical, financial, military data ...)

- **R**eproducibility

- How easy is it to get the attack to work in the wild?
- Would the attack use a feature installed by default?

- **E**xploitability

- How much effort and expertise is required to mount an attack?
- Can the attack be scripted (automation)?
- Anonymous remote user vs. local user with strong credentials

# DREAD

- Affected Users

- How many users would be affected? Not just percentage
- Does the attack affect clients or servers?
- Does the attack only work in special configurations?

- Discoverability

- Will the vulnerability be discovered by potential attackers?
- The safe option is to assume that it will (10/10)

➤ Should cost and effort to mitigate the threat be considered in ranking threats?

# MS11-083 – Exploitability

0	Exploitation detected
1	Exploitation more likely *
2	Exploitation less likely **
3	Exploitation unlikely ***

- Reference counter issue in TCP/IP stack
- Send large number of specially crafted UDP packets to a port that does not have a service listening
- Dereferencing a data structure can cause remote code execution (RCE)
- “We (MS) believe it is difficult to achieve RCE using this vulnerability considering that the type of network packets required are normally filtered at the perimeter and the small timing window between the release and next access of the structure, and a large number of packets are required to pull off the attack.”
- **Exploitability Index** “2” assigned for this vulnerability

<https://blogs.technet.microsoft.com/srd/2011/11/08/assessing-the-exploitability-of-ms11-083/>



# Computing Risks

- For each category, assign a value between 1 and 10
  - How you actually do it depends on experience and conventions you may have adopted
- Average over all five values gives the **risk rating**
- E.g.: unauthorized user reads payroll data on network
  - 8 Damage potential: serious privacy intrusion
  - 10 Reproducibility: 100% reproducible
  - 7 Exploitability: user must be on the subnet
  - 10 Affected Users: everyone is affected
  - 10 Discoverability: assume people will find out
- **Overall result: risk = 9**

# Comments

- Remember, risk analysis is not an exact science
- Picking numbers in DREAD (or any other methodology for that matter) is to some degree arbitrary
- You need guidelines to get **consistent** ratings
- You need experience to know when you are getting wrong results; you can then adjust your rating scheme accordingly

# Cumulative Voting

- In a method like DREAD there is a temptation to give high ratings all the time
- This is not helpful for prioritizing risks
- **Cumulative voting** (\$100 method) can help avoiding this problem:
  - You have a fixed number of 'votes' (or dollars) you can assign to the risks you have identified
  - You can give more than one vote to a risk
  - You must be clear about your **priorities** when casting votes
- Used, e.g., in software development at Ericsson

# CVSS: Common Vulnerability Scoring System

- CVSS starts from the vulnerabilities when organizing impact assessment
  - <http://www.first.org/cvss/>
  - <http://web.nvd.nist.gov/view/vuln/search> (US National Vulnerability Database)
- Produce a numerical score reflecting its severity
- Impact of a vulnerability **may change over time**
- Impact of a vulnerability may depend on the **specific environment** a system is deployed in
- Not all sources of vulnerability reports are equally reliable

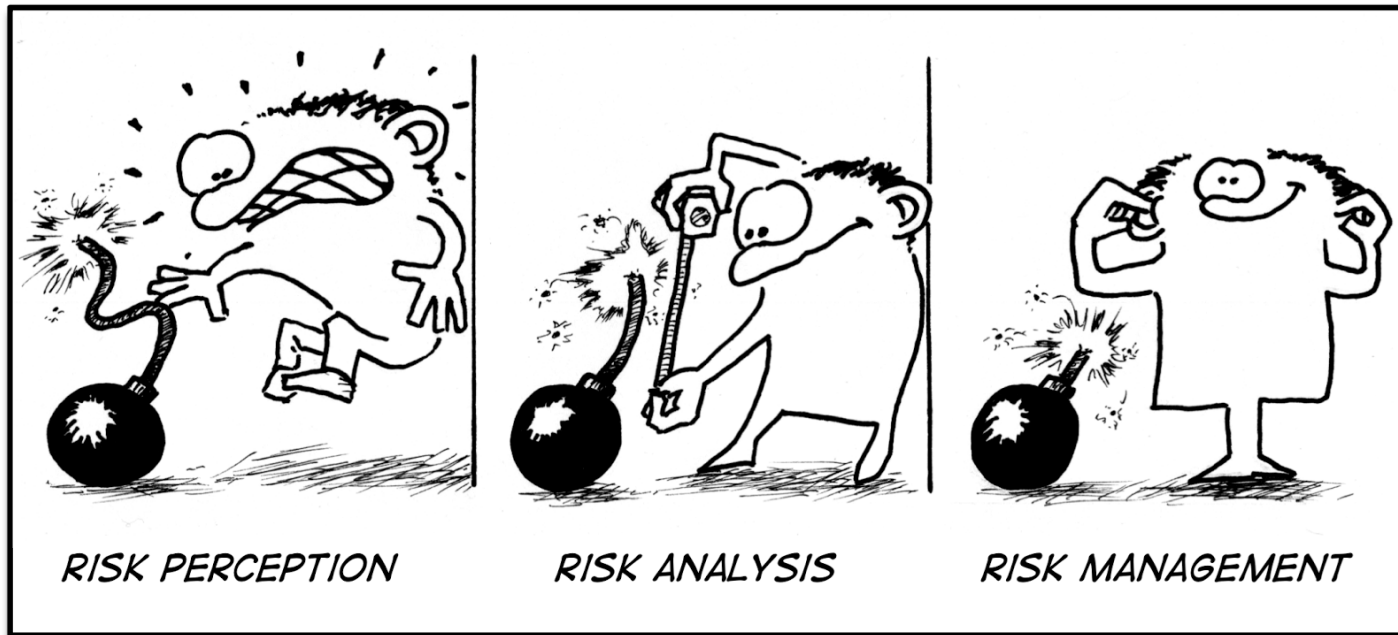
# CVSS – Scoring Scheme

Basic metrics		Temporal metrics	Environmental metrics	
Attack vector	Confidentiality impact	Exploit code maturity (exploitability)	Collateral damage potential	Confidentiality requirement
Attack complexity	Integrity impact	Remediation level	Target distribution	Integrity requirement
Privilege Required	Availability impact	Report confidence		Availability requirement

From these individual ratings the **CVSS severity** is computed (more details in tutorial)

# Brining It All Together: Decomposition, Threat Trees, and Ranking Mechanism

- To bring it all together, you can determine the **threat targets** from functional decomposition, determine **types of threat** to each component by using STRIDE, use threat trees to determine **how the threat can become a reality**, and apply a **ranking mechanism** (DREAD, CVSS), to each threat
- To apply STRIDE for each threat target, ask:
  - Is this item susceptible to spoofing?
  - Can this item be tampered with?
  - Can an attacker repudiate this action?
  - Can an attacker view this item?
  - Can an attacker deny service to this process or data flow?
  - Can an attacker elevate their privilege by attacking this process?



# Mitigate Threats

Choose how to respond to the threats

# Responding to Threats

- Do nothing
  - May only delay the pain; when somebody else finds out about the problem you will have to respond
  - If removing the problem would break an important application, you may be better off doing nothing
- Warn the user
  - Tell the users that they are about to do something dangerous and let them decide whether to go ahead
  - Users are rarely qualified to make such decisions



# Responding to Threats Cont'd

- Remove the problem
  - Remove dangerous feature from the product
  - There's always the next version
- Fix the problem
  - Remedy the problem with technology

# Choose Techniques to Mitigate the Threats

- Techniques are not the same as technologies
  - A technique is derived from a high-level appreciation of what kinds of technologies can be applied to mitigate a threat
  - For example, authentication is a security technique, and Kerberos is a specific authentication technology

# Example of Mitigation Techniques

Threat Type	Mitigation Techniques
Spoofing identity	Appropriate authentication, Protect secret data
Tampering with data	Appropriate authorization, Hashes, MACs, Digital signatures, Tamper-resistant protocols
Repudiation	Digital signatures, Timestamps, Audit trails
Information disclosure	Authorization, Privacy-enhanced protocols, Encryption, Protect secrets
Denial of service	Appropriate authentication, authorization, Filtering, Throttling, Quality of service
Elevation of privilege	Run with least privilege

# Choose the Appropriate Technologies for the Identified Techniques

Choose the one suitable for your domain-specific problem. For example, to avoid spoofing, authentication technique can be applied; **choose the technology that suits your authentication requirement:**

Protocol	Authenticate Client?	Authenticate Server?
Basic	Yes	No
Digest	Yes	No
Forms	Yes	No
Kerberos	Yes	Yes
X.509 certificates	Yes	Yes
IPSec	Yes (computer)	Yes (computer)

# Example: Mitigating the Sample Payroll Application Threats

Threat	STRIDE	Techniques & Technologies
View or manipulate payroll data	I & T	Use SSL/TLS to encrypt the channel between the server & client. Could also use IPSec
Upload rouge web pages	T	Use strong authentication for the web developers. Provide strong ACLs on the files
Perform DoS	D	Use a firewall to drop certain IP packets. Restrict resources used by anonymous users (e.g. memory, database time).
Elevate privilege by leveraging the service client request process	E	Run the process following the guidelines of least privilege so even if the process is compromised, extra capabilities are not gained

# Items to Note

- Identify all possible **threat targets**; consider all **interfaces** to the system regardless of whether they are published
- How would an attacker manifest the threat? Keep the threat trees simple
- Risk: use preferred method of calculating risk. Make sure to be consistent
- **Mitigation status**: has the threat been mitigated? Yes, No, Somewhat, and Needs Investigation!
- **Bug-tracking**: utmost important to have a bug-tracking database and make sure it is up-to-date

# Attack Surface Analysis

Entry/Exit Points of Attacks

# Attack Surfaces

- The attack surface of a software environment is the sum of the different points (the “**attack vectors**”), where an attacker can try to enter data to or extract data from an environment
- Keeping the attack surface as small as possible is a basic security measure
- Compared with **threat modelling**:
  - Add details of entry/exit points of attacks in the system when determining threats to components identified after decomposing the system
  - Reduce attack surface during the mitigation step



# Attack Surfaces

- Three dimensions of attack surface: **targets & enablers, channels & protocols, access rights**
- “**Attackability**”: measure of how exposed a system’s attack surface is
- Reduce attack surface: eliminate or reduce number of
  - types or instances of **targets**, processes, enablers, carriers, channels, protocols, and rights
  - types or instances of **attacks**, e.g., through deploying one or more security technologies

# Attack Vectors for Windows

1. Open sockets: TCP or UDP sockets on which at least one service is listening
2. Open RPC endpoints: Remotely-accessible handlers registered for remote procedure calls
3. Open named pipes: Remotely-accessible named pipes on which at least one service is listening
4. Services: Services installed, but not disabled
5. Services running by default
6. Services running as SYSTEM
7. Active Web handlers
8. Active ISAPI filters: ISAPI filter is a dynamic link library (.dll) that uses ISAPI to respond to events that occur on the server
9. Dynamic web pages

# Attack Vectors for Windows

10. Executable vdirs: “Virtual Directories”
11. Enabled accounts
12. Enabled accounts in admin group
13. Null sessions to pipes and shares
14. Guest account enabled
15. Weak ACLs in FS
16. Weak ACLs in Registry
17. Weak ACLS on shares
18. VBScript (Visual Basic Script ) enabled
19. Jscript enabled
20. ActiveX enabled

# Computing Attack Surfaces

- **Attack surface area:** sum of independent weighted contributions from:
  - a set of channels types,
  - a set of process target types,
  - a set of data target types,
  - a set of process enablers,
  - all subject to the constraints of the access rights relation
- Gives you a unified number
- Pragmatic approach; suited reasonably well **for comparing closely related systems**, e.g. different versions of the same OS

# Summary

# Summary

- Risk and threat analysis is organized common sense
- Guidance to support consistent rating is important
- There are a number of different approaches; pick the one that suits you best; there is no correct answer!
- Attack surfaces do not measure security, rather exposure
- The search for good security metrics is still an active research area

# Summary

- It's important that you spend time in this phase
- After all, it's cheaper to find a security design bug at this stage and remedy the solution before coding starts
- You must also keep the threat model current, reflecting new threats and mitigations as they arise
- Threat models are also useful for testers, for designing tests against the threat model (for traceability and to demonstrate compliance)

# Tutorial – Tuesday next week

1. Look more closely at CVSS, discuss whether this is ‘qualitative risk analysis’ (inputs are not numbers but choices from menus, output is low-medium-high) or ‘quantitative risk analysis’ (internally, numbers are used)
2. CVSS v2 versus CVSS v3
  - <https://www.first.org/cvss/examples>
  - [https://www.first.org/cvss/cvss-v30-examples\\_v1.1.pdf](https://www.first.org/cvss/cvss-v30-examples_v1.1.pdf)
3. CVSS for XSS (why is XSS a medium risk?)
  - <https://nvd.nist.gov/vuln/detail/CVE-2016-2163>
  - <https://www.cvedetails.com/cve/CVE-2017-7271/>