Security Engineering

Applied Information Security
Summer 2021, Lecture 5

Course so far

attackers

- mindset, phases, **attacks**.
  - recon/scan/access ← **tools** for this

sys-admins

- **isolation** firewalls, containers, VMs
- **audit** intrusion detection, vulnerability scan, antivirus, …

Course so far

attackers

- mindset, phases, **attacks**.
  - recon/scan/access ← **tools** for this

sys-admins

- **isolation** firewalls, containers, VMs
- **audit** intrusion detection, vulnerability scan, antivirus, …

(great, but)
often not possible (sharing), often cumbersome

(great, but)
what is good/bad behavior? security “too late”

security is **application-specific** (which ops are OK?).
how to **specify** & **enforce** app-specific security concerns?

Course so far

attackers

- mindset, phases, **attac**
  - recon/scan/access ←

sys-admins

- **isolation** firewalls
- **audit** intrusion d tion,
  vul rability scan, antivirus, …

security is **application-specific** (which ops are OK?).
how to **specify** & **enforce** app-specific security concerns?

spoiler:

we don't know!

story time, courtesy of Butler Lampson

(great, but)
often not possible (sharing),
often cumbersome

(great, but)
what is good/bad behavior?
security "too late"

Why is it hard?

in the beginning: security = physical isolation. 1950-1963

bring data, control machine, take everything away. **Easy**

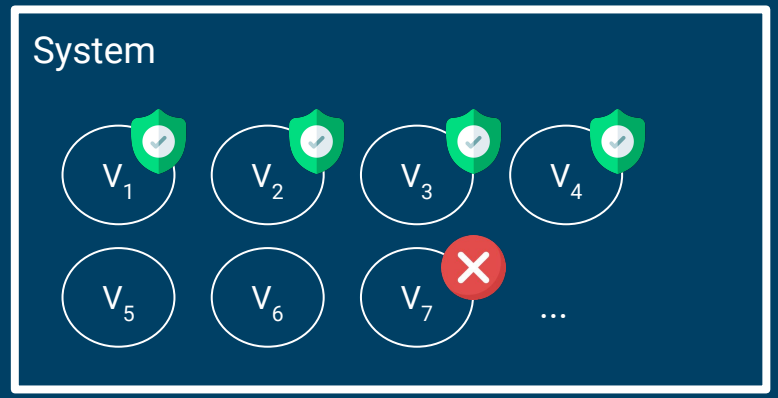
time-sharing brought security dilemma: **isolation vs. sharing** 1963-1982

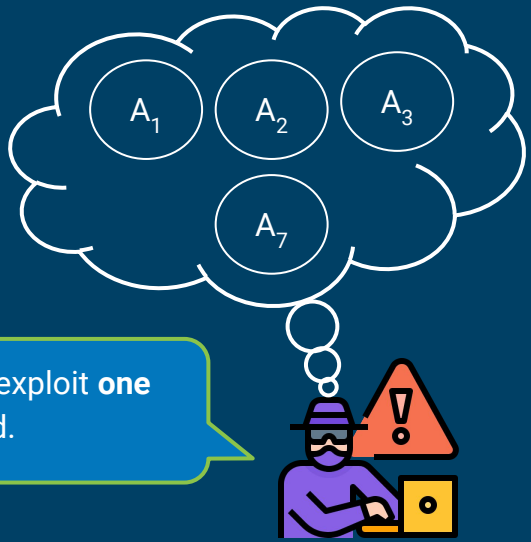
each user wants private machine, isolated from others.
users want to share data, programs, resources. **Hard**

since then, things have only gotten worse 1982-today

less isolation, more sharing, no central management, more valuable data, continued misguided search for perfect security, ...

Why is it hard? - **disparity**





security engineer must protect against **all** vulnerabilities.

attackers only need to exploit **one** vulnerability to succeed.

Security Engineer

virtually impossible. vulnerabilities might not even be known.

Attacker

Where we stand

What we can do

- secure something simple very well
- protect complexity (isolation, sanitize)
- stage security theater :-P

What we can't do

- make something complex secure
- make something big secure (if not isolated)
- keep something secure when it changes
- get users to make judgements about security
- understand privacy

we have learned a lot of valuable lessons & tricks, though.
those are the focus of today!

Today

wisdom of the sages (through the ages).

- security **principles** — best practises
- security **mechanisms** — technical solutions
- security **requirements** — how to specify good behavior
- security **evaluation** — expectations (by gov/ind.)

Security Principles

Guidelines; Stood the Test of Time

Principles

- Complete Mediation
- Failsafe Defaults
- Least Privilege
- Separation of Privilege
- Open Design
- Defense in Depth
- Psychological Acceptability
- Isolation
- Minimum Exposure
- Least Common Mechanism
- Accountability

Common Theme

- **separation**
- **redundancy**
- **simplicity**
- **completeness**

make note!

Source

- Saltzer & Schroder 1975
- Butler W. Lampson
- Fred B. Schneider

not an exhaustive list

Complete Mediation

monitor and control every operation to every object by every principal.

intercept the action (e.g. access, write), determine if operation wrt. policy.

implications:

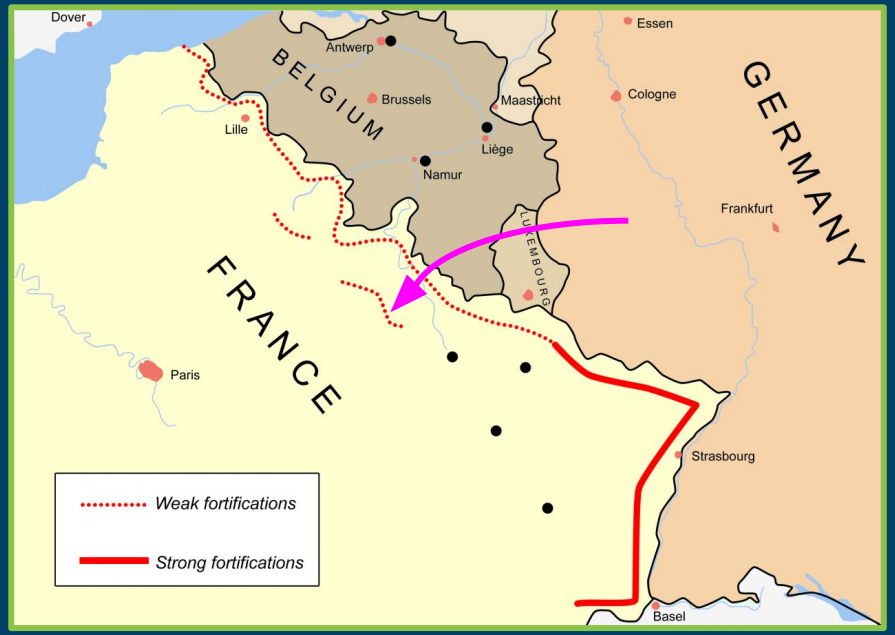
- system-wide access control
- fool-proof way to ID source of req.
- restrictions on caching

primary underpinning of protection.

Complete Mediation

monitor and control every operation to every object by every principal.

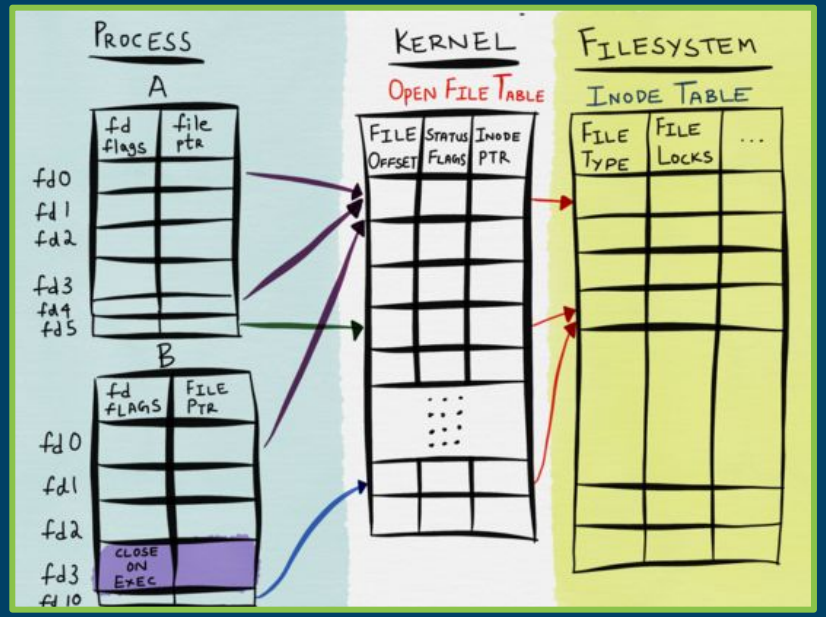
Maginot-line: Strong fortifications didn't extend all the way (WW2).



Complete Mediation

monitor and control
every operation to
every object by
every principal.

UNIX mediates all file system access;
process tries to read a file
⇒ kernel decides if process is allowed.



Complete Mediation

monitor and control every operation to every object by every principal.

… unless accessed physically (physical access bypasses the kernel) (countermeasure: disk encryption).

Failsafe Defaults

access should be denied by default, and only granted explicitly (by mechanism).

example: mail server cannot create file in `/var/spool`. store elsewhere?

Failsafe Defaults

access should be denied by default, and only granted explicitly (by mechanism).

example: mail server cannot create file in `/var/spool`. store elsewhere?

attacker can read the mail there.
attacker can fill more hard drives (DoS).
attacker can get root (privilege escalation).

example: access card system down. grant card-bearers access?

Failsafe Defaults

access should be denied by default, and only granted explicitly (by mechanism).

example: mail server cannot create file in `/var/spool`. store elsewhere?

attacker can read the mail there.
attacker can fill more hard drives (DoS).
attacker can get root (privilege escalation).

example: access card system down. grant card-bearers access?

heist of 500 engine parts from a German car manufacturer

Failsafe Defaults

access should be denied by default, and only granted explicitly (by mechanism).

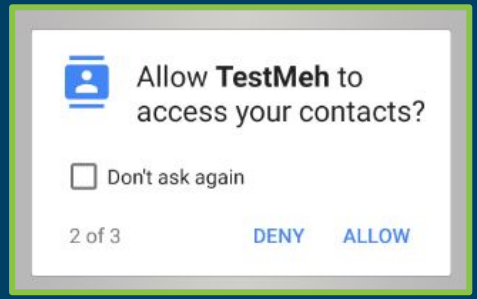
example: browsers & TLS

machine does not want to talk to you
in a manner you're happy with (TLS 2)?
don't talk to them insecurely;
just don't talk to them.

Failsafe Defaults

access should be denied by default, and only granted explicitly (by mechanism).

example: Android runtime permissions



denies access by default.
requires explicit permission from users.

more a **safe** default than a **fail**safe default

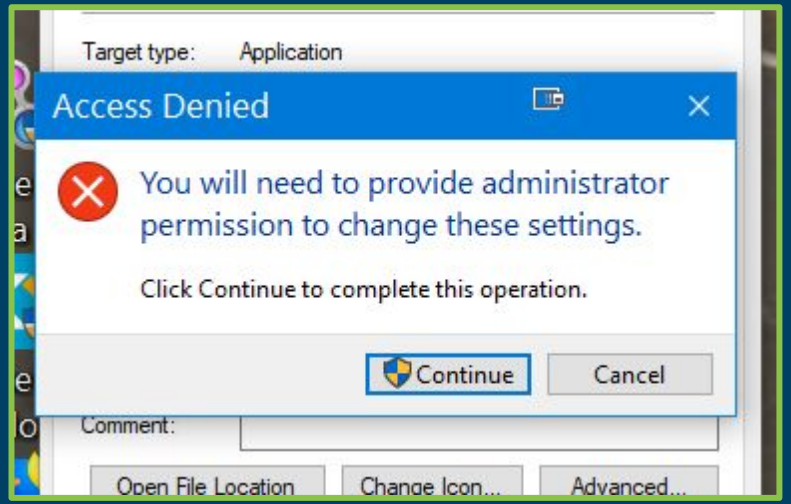
Least Privilege

principals should operate with least set of privileges needed to complete operation

does it *really* need to run as `root`?

- computer game?
- PDF viewer?
- WinZIP?

Would you hit "Continue"?



Least Privilege

principals should operate with least set of privileges needed to complete operation

limits damage / malice resulting from improper use of privilege less likely.

- justifies “need to know”.

UNIX: my process does not have root
⇒ it can only only leak / destroy my files (not whole sys, other users) if it “goes rogue”.

Separation of Privilege

different operations require different privileges.

split program into separate processes with their own privileges.

- one process compromised ⇒ less damage (e.g. just a DoS)
- communication between parts goes through OS ⇒ security check

major feature of OpenBSD

Open Design

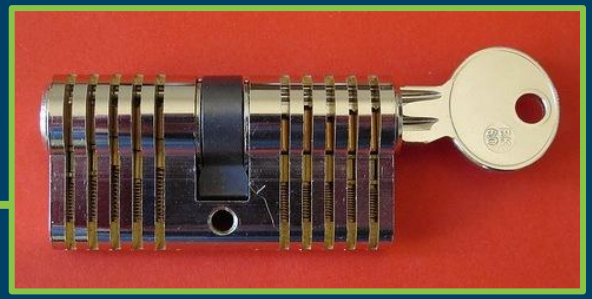
security should not depend on the protection mechanism being secret

Kerckhoff's Principle, 1883:

"security through obscurity" bad idea.

- don't depend on attacker ignorance
  - hard to control what they know
  - (dumpster dive, phish, reverse engineer, ...)
  - 93% of modern Web App code is **OS** libs...
- depend on possession of keys/passwd
  - easy to protect

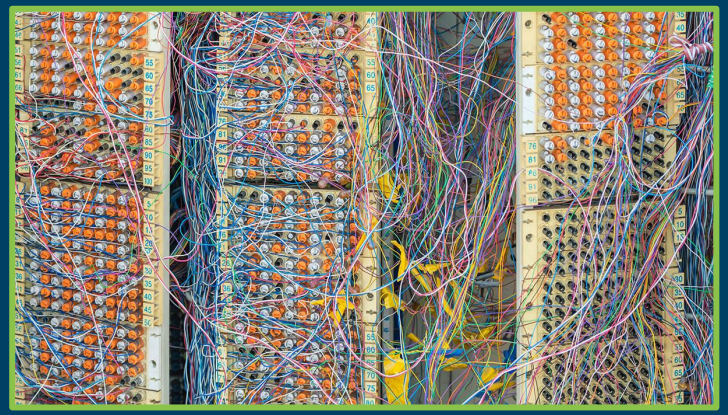
even if attacker knows algorithms, we *still* have assurance.



Economy of Mechanism

mechanism must be as simple as possible.

complex design ⇒ complex failures



simpler security mechanism ⇒

- fewer errors
- smaller TCB

Defense in Depth

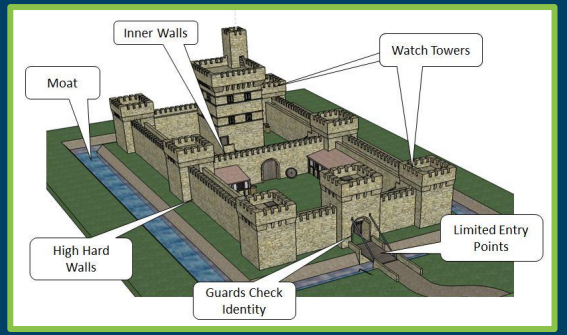
use a set of
independent and overlapping
mechanisms,
instead of a single mechanism
no single point of failure

no single mechanism resist all attacks.

- separation of duty
- redundancy:
  no single point of failure

example: ATM (card + password)

example: e-mail (firewall + sandbox)



Defense in Depth

use a set of
independent and overlapping
mechanisms,
instead of a single mechanism
no single point of failure

example: two-factor authentication

Psychological Acceptability

mechanism must not make resources more difficult use than if mechanism not present

security must be *usable* (else circumvented)

How many of you would just hit "agree" (Enig)?

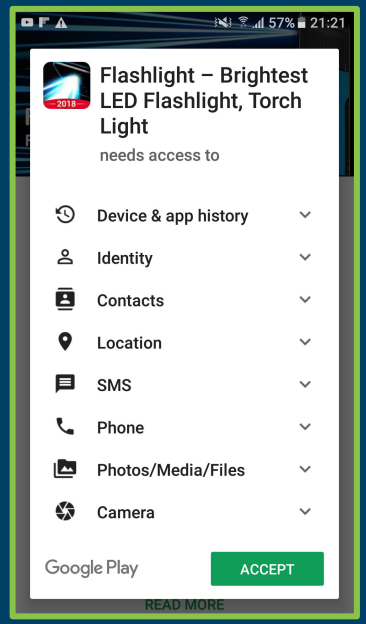
Isolation

organize resources into
isolated groups of similar needs

mechanism that implements this principle: firewall, VM, …

A & B don't need to communicate?
don't enable them to communicate.

- contain failure



Would you hit "ACCEPT"?

Isolation

organize resources into
isolated groups of similar needs

mechanism that implements this principle: firewall, VM, …

A & B don't need to communicate? don't enable them to communicate.

- contain failure

example: browser. sandboxed; tabs cannot interact, JS can't access disk

example: don't run your IDE, browser, etc. on the server that hosts your company's valued database.

example: in fact, don't make that server directly reachable from the Internet.

Minimum Exposure

minimize attack surface that the system presents to attacker

reduce external interface.

don't need it? turn it off!

service on a port,

device on the network, …

think IoT

the less SW you run, the safer you are.

Least Common Mechanism

means of
accessing a resource
should not be shared

sharing may lead to *vulnerabilities*.

- hardware
- OS/software
- *mechanism*

example: DoS attack on PayPal ⇒ companies can't be paid
(PayPal shared by companies & attacker)

Accountability

hold principals
legally responsible
for their actions.

mechanism that implements this principle: logging, intrusion detection.

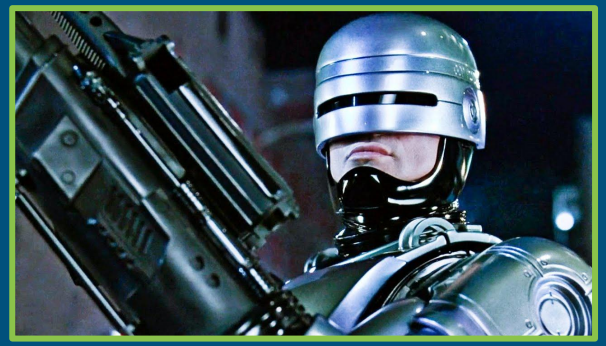
we can’t achieve perfect security yet.

alternative: accountability

- values
- locks
- punishment

complementary; disincentivize attacks.

Security Mechanisms



basic mechanisms

for implementing security

Gold Standard

Butler W. Lampson,
Turing Award winner 1992

authenticate principals

- “Who said that?”
- “Who is getting that information?”

authorize access

- “Who can do which operation on which object?”

audit decision of guard

- “What happened? Why?”
  Not always an option! (Voting)

Mechanisms - Gold Standard

Authenticate

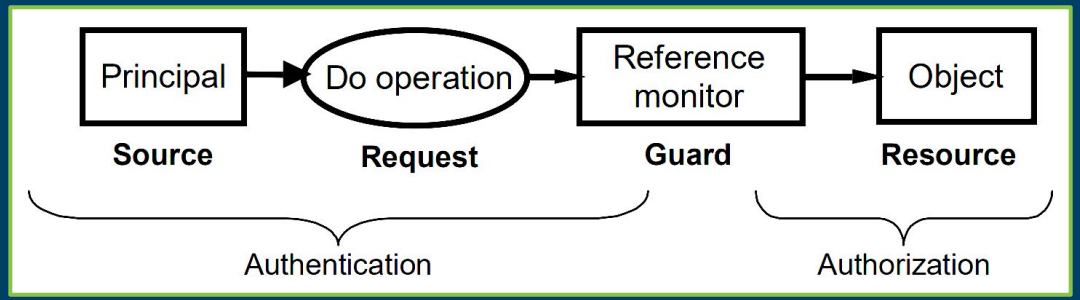
determine whom you are to the system.

- identification
  - **indicate** identity (from observed attributes)
  - **example:** surveillance cameras looking for an individual in a crowd
- authentication
  - **verify** identity (proof)
  - **example:** a security officer at border control verifying that a passport belongs to its bearer

Examples of proof: password, token, fingerprint

Authorize: Access Control

Guard decides if principal is allowed to do operation on object.



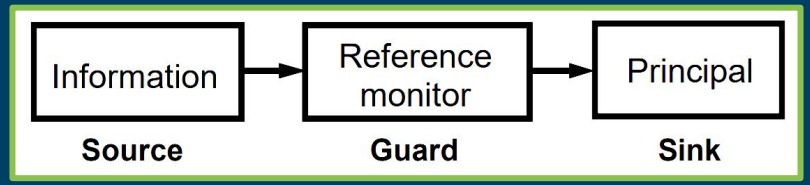
authentication: identify the principal who made the request.

authorization: who can do this operation on that object?

why separate guard from object: simplicity. (smaller TCB).

Authorize: Information-Flow Control

Guard decides if information can flow to principal.



authentication: identify the principal who receives the information.

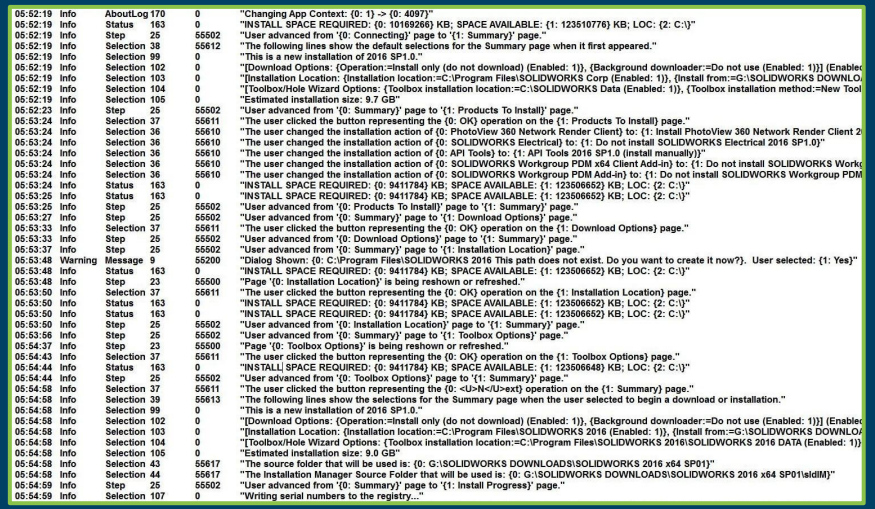
authorization: who can receive this information?

Mechanisms - Gold Standard

Auditing

security log: security-relevant event

- access, authentication failure, etc.
- provides *audit trail*.



05:52:19	Info	AboutLog	170	0	"Changing App Context: {0: 1} -> {0: 4097}"
05:52:19	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 10169266} KB; SPACE AVAILABLE: {1: 123510776} KB; LOC: {2: C:\}"
05:52:19	Info	Step	25	55502	"User advanced from '{0: Connecting}' page to '{1: Summary}' page."
05:52:19	Info	Selection	38	55612	"The following lines show the default selections for the Summary page when it first appeared."
05:52:19	Info	Selection	99	0	"This is a new installation of 2016 SP1.0."
05:52:19	Info	Selection	102	0	"[Download Options: {Operation:=Install only (do not download) (Enabled: 1)}, {Background downloader:=Do not use (Enabled: 1)}] (Enable
05:52:19	Info	Selection	103	0	"[Installation Location: {Installation location:=C:\Program Files\SOLIDWORKS Corp (Enabled: 1)}, {Install from:=G:\SOLIDWORKS DOWNLO
05:52:19	Info	Selection	104	0	"[Toolbox/Hole Wizard Options: {Toolbox installation location:=C:\SOLIDWORKS Data (Enabled: 1)}, {Toolbox installation method:=New Tool
05:52:19	Info	Selection	105	0	"Estimated installation size: 9.7 GB"
05:52:23	Info	Step	25	55502	"User advanced from '{0: Summary}' page to '{1: Products To Install}' page."
05:53:24	Info	Selection	37	55611	"The user clicked the button representing the {0: OK} operation on the {1: Products To Install} page."
05:53:24	Info	Selection	36	55610	"The user changed the installation action of {0: PhotoView 360 Network Render Client} to: {1: Install PhotoView 360 Network Render Client 2
05:53:24	Info	Selection	36	55610	"The user changed the installation action of {0: SOLIDWORKS Electrical} to: {1: Do not install SOLIDWORKS Electrical 2016 SP1.0}"
05:53:24	Info	Selection	36	55610	"The user changed the installation action of {0: API Tools} to: {1: API Tools 2016 SP1.0 (install manually)}"
05:53:24	Info	Selection	36	55610	"The user changed the installation action of {0: SOLIDWORKS Workgroup PDM x64 Client Add-in} to: {1: Do not install SOLIDWORKS Work
05:53:24	Info	Selection	36	55610	"The user changed the installation action of {0: SOLIDWORKS Workgroup PDM Add-in} to: {1: Do not install SOLIDWORKS Workgroup PDM
05:53:24	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 9411784} KB; SPACE AVAILABLE: {1: 123506652} KB; LOC: {2: C:\}"
05:53:25	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 9411784} KB; SPACE AVAILABLE: {1: 123506652} KB; LOC: {2: C:\}"
05:53:25	Info	Step	25	55502	"User advanced from '{0: Products To Install}' page to '{1: Summary}' page."
05:53:27	Info	Step	25	55502	"User advanced from '{0: Summary}' page to '{1: Download Options}' page."
05:53:33	Info	Selection	37	55611	"The user clicked the button representing the {0: OK} operation on the {1: Download Options} page."
05:53:33	Info	Step	25	55502	"User advanced from '{0: Download Options}' page to '{1: Summary}' page."
05:53:37	Info	Step	25	55502	"User advanced from '{0: Summary}' page to '{1: Installation Location}' page."
05:53:48	Warning	Message	9	55200	"Dialog Shown: {0: C:\Program Files\SOLIDWORKS 2016 This path does not exist. Do you want to create it now?}. User selected: {1: Yes}"
05:53:48	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 9411784} KB; SPACE AVAILABLE: {1: 123506652} KB; LOC: {2: C:\}"
05:53:48	Info	Step	23	55500	"Page '{0: Installation Location}' is being reshown or refreshed."
05:53:50	Info	Selection	37	55611	"The user clicked the button representing the {0: OK} operation on the {1: Installation Location} page."
05:53:50	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 9411784} KB; SPACE AVAILABLE: {1: 123506652} KB; LOC: {2: C:\}"
05:53:50	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 9411784} KB; SPACE AVAILABLE: {1: 123506652} KB; LOC: {2: C:\}"
05:53:50	Info	Step	25	55502	"User advanced from '{0: Installation Location}' page to '{1: Summary}' page."
05:53:56	Info	Step	25	55502	"User advanced from '{0: Summary}' page to '{1: Toolbox Options}' page."
05:54:37	Info	Step	23	55500	"Page '{0: Toolbox Options}' is being reshown or refreshed."
05:54:43	Info	Selection	37	55611	"The user clicked the button representing the {0: OK} operation on the {1: Toolbox Options} page."
05:54:44	Info	Status	163	0	"INSTALL SPACE REQUIRED: {0: 9411784} KB; SPACE AVAILABLE: {1: 123506648} KB; LOC: {2: C:\}"
05:54:44	Info	Step	25	55502	"User advanced from '{0: Toolbox Options}' page to '{1: Summary}' page."
05:54:58	Info	Selection	37	55611	"The user clicked the button representing the {0: <U>N</U>ext} operation on the {1: Summary} page."
05:54:58	Info	Selection	39	55613	"The following lines show the selections for the Summary page when the user selected to begin a download or installation."
05:54:58	Info	Selection	99	0	"This is a new installation of 2016 SP1.0."
05:54:58	Info	Selection	102	0	"[Download Options: {Operation:=Install only (do not download) (Enabled: 1)}, {Background downloader:=Do not use (Enabled: 1)}] (Enable
05:54:58	Info	Selection	103	0	"[Installation Location: {Installation location:=C:\Program Files\SOLIDWORKS 2016 (Enabled: 1)}, {Install from:=G:\SOLIDWORKS DOWNLO
05:54:58	Info	Selection	104	0	"[Toolbox/Hole Wizard Options: {Toolbox installation location:=C:\Program Files\SOLIDWORKS 2016\SOLIDWORKS 2016 DATA (Enabled: 1)}
05:54:58	Info	Selection	105	0	"Estimated installation size: 9.0 GB"
05:54:58	Info	Selection	43	55617	"The source folder that will be used is: {0: G:\SOLIDWORKS DOWNLOADS\SOLIDWORKS 2016 x64 SP01}"
05:54:58	Info	Selection	44	55617	"The Installation Manager Source Folder that will be used is: {0: G:\SOLIDWORKS DOWNLOADS\SOLIDWORKS 2016 x64 SP01\sldIM}"
05:54:59	Info	Step	25	55502	"User advanced from '{0: Summary}' page to '{1: Install Progress}' page."
05:54:59	Info	Selection	107	0	"Writing serial numbers to the registry..."

provenance: full program behavior

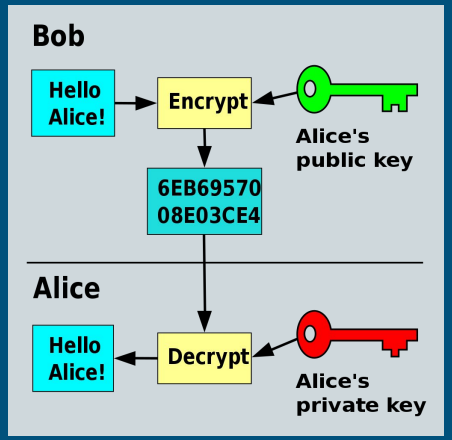
- detect anomalous behavior

intrusion detection: monitor network or system for policy violations

techniques

for implementing gold standard

Cryptography

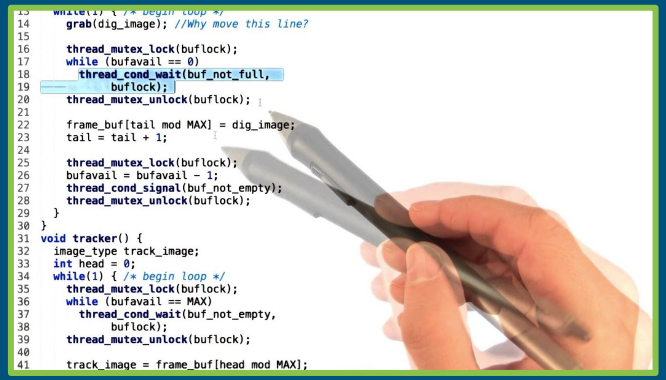


scramble data, so original data can only be read if you possess the key.

facilitates secure communication over untrusted medium (e.g. Internet).

- confidentiality (encryption)
- integrity (signature)

Program Analysis



scan the code w/o running it,
to determine if it e.g.

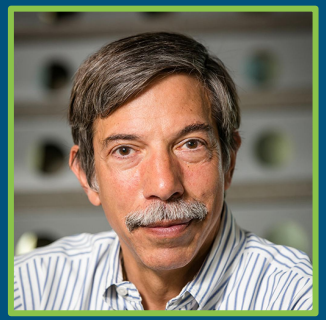
- has vulnerabilities
- satisfies security requirements

(think “linters”, “type systems”, etc.)

example: SpotBugs for Java
(exercise session)

Monitors

Fred B. Schneider
extensive work on monitors



monitor interface I/O. halt execution before damage is done.

what's needed:

- **policy:** which I/O is okay
- **monitor:** receives control upon I/O
- ability to **block** program

white/black-box, inlined, ...

Isolation

restrict / prevent communication.

process: executes in its own address space. access to shared resources.

sandbox: provides “shadow copy” in response to request to environment.

virtual machine: computer simulated in software. limited access to host.

firewall: pass certain traffic through.

A Caveat

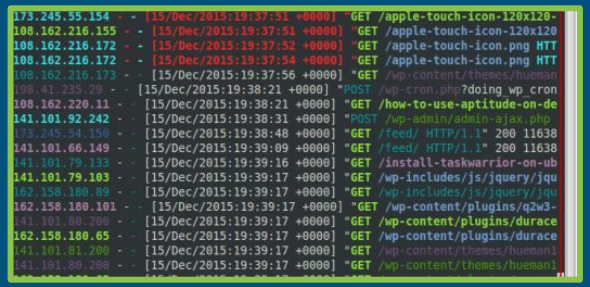
> “Isolation plays the same role in computer security as did the tall, imposing perimeter walls in protecting a medieval city from marauders. [...] Note the tension between defending the city, & promoting daily activities of citizens”
>
> - Fred B. Schneider

isolation vs. security

when useful:

- little pressure to puncture boundaries
- communication that does cross boundary is limited & carefully prescribed.

Logging



actions; how system came to be

What to log: **events**, e.g.

- login
- access to protected resource,
- elevation of privileges, ...

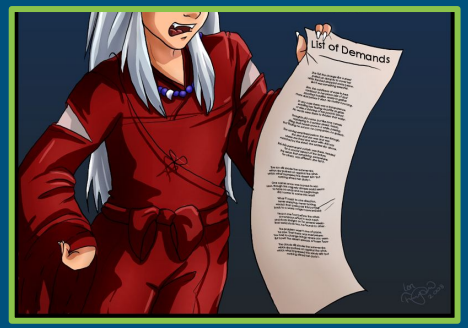
which events? **security-relevant** ones.

what to put in the log entry?

- what check was made,
- outcome
- information that lead to that decision.

facilitates **auditing**

Security Requirements



System-Specific Security Requirements

what ***should*** the system do?

security a system is secure iff it does what it *should*, and nothing more.

engineering methodology to arrive at **security requirements**:

1. functional requirements
2. threat analysis
3. harm analysis
4. security goals
5. feasibility analysis
6. security requirements

finally, look at existing methodologies (aka. **threat modeling** methodologies)

functional requirements

Rules of Thumb

functional requirement: *specification* of *behavior*, between outputs & inputs of system (or component).

we start with

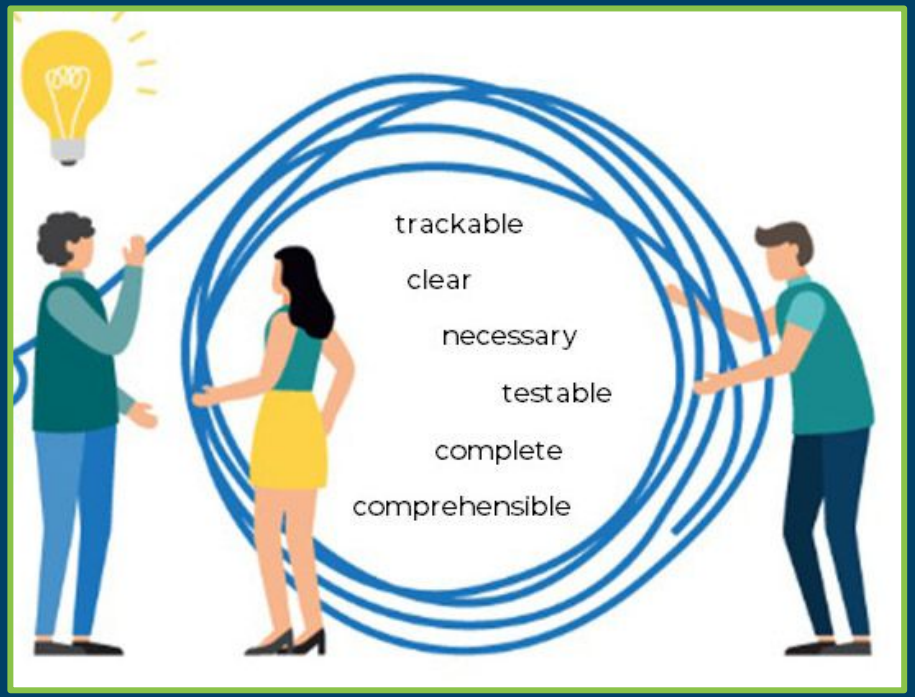
functional requirements

we end with

security requirements

both expected to satisfy. →

today: how to arrive at these.



User Story

brief description of single kind of interaction user can have w/ system

format:

"as a **[** user **]**,
I can **[** action **]**
so that **[** benefit **]**."

goal

reason, value

user stories reveal system **assets**.

Example User Story

Example (course CMS)

- “As a professor, I can create a new assignment by specifying its name, possible grades, and due date.”
- “As a student, I can submit a file as a solution to an assignment.”

Example User Story

Example (course CMS)

- “As a professor, I can create a new assignment by specifying its name, possible grades, and due date.”
- “As a student, I can submit a file as a solution to an assignment.”

threat analysis

Rules of Thumb

identify threats of concern to system

- especially **malicious**, **human** threats
- what kinds of attackers will system resist?
- what are their motivations? resources? capabilities?

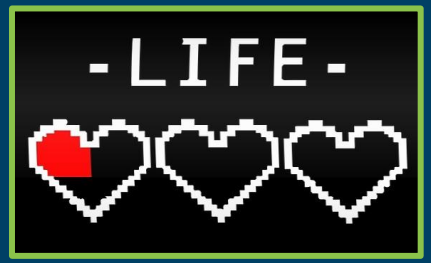
best if analysis is specific to system and its functionality

non threats:

- trusted hardware, trusted environment
  e.g., physically secured machine room reachable only by trustworthy system operators

harm analysis

Rules of Thumb



harm: action adversely affects value of asset.

harm to…

- **confidentiality**: disclosure
- **integrity**: modification or fabrication
- **availability**: deprivation / loss-of-use

format:

- “Performing [*action*] { on, to, with } [*asset*] could cause [*harm*].”
  e.g., "stealing money could cause loss of revenue"
  e.g., "erasing account balances could cause loss of customers"

Harm Triples

⟨ *action*, *asset*, *harm* ⟩

examples:

- ⟨theft, money, lose revenue⟩
- ⟨erasure, account balance, lose customer⟩

methodology:

- start with an **asset**
- brainstorm **actions** that could **harm** that asset

let brainstorming be guided by CIA

Example: Harm Triples

Grade Management System (GMS). Manages just the final grade for one course.

functional requirements:

- “as a *student*, I can view my final grade.”
- “as a *professor*, I can view and change final grades for all students.”
- “as an *administrator*, I can add/remove students/professors to/from course.”

Example: Harm Triples

Grade Management System (GMS). Manages just the final grade for one course.

functional requirements:

- “as a *student*, I can view my final grade.”
- “as a *professor*, I can view and change final grades for all students.”
- “as an *administrator*, I can add/remove students/professors to/from course.”

Example: Harm Triples

Grade Management System (GMS). Manages just the final grade for one course.

threat analysis:

- students:
  - **motivations**: increase their own grade, lower others' grades, learn others' grades
  - **capabilities**: network access to servers, some physical access to others' computers, social engineering; probably not extensive computational or financial resources
- out of scope:
  - assume that threats cannot physically access any servers
  - professors are trusted, system admins are trusted

Example: Harm Triples

Grade Management System (GMS). Manages just the final grade for one course.

asset: grade for each student.

functional requirement: students view grade, profs view/change grade, admins manage enrollment

threat analysis: malicious/curious students.
professors trusted. no physical access.

<u>in class exercise:</u>

harm analysis: "Performing **[** *action* **]** **{** on, to, with **}** **[** *asset* **]** could cause **[** *harm* **]**."

⟨ *action*, *asset*, *harm* ⟩ ← invent some!

Example: Harm Triples

Grade Management System (GMS). Manages just the final grade for one course.

asset: grade for each student.

functional requirement: students view grade, profs view/change grade, admins manage enrollment

threat analysis: malicious/curious students.
professors trusted. no physical access.

<u>in class exercise:</u>

harm analysis: "Performing **[** *action* **]** **{** on, to, with **}** **[** *asset* **]** could cause **[** *harm* **]**."

⟨ *disclosure*, *grade*, *embarrassment / loss of employability* ⟩ ← **confid.**

⟨ *overwriting*, *grade*, *GPA is lowered* ⟩ ← **integrity**

⟨ *spam-request*, *grade*, *grade is unviewable* ⟩ ← **availability**

security goals

also known as:

Security Policy

security a system is secure iff it

- does what it *should*,
- and nothing more.

policy stipulates what should and should not be *done*.

format “The system shall
{ prevent, detect } [*action*]
{ on, to, with } [*asset*].”

how turn ⟨ *action*, *asset*, *harm* ⟩ into above format.

Examples

format “The system shall { prevent, detect } [*action*] { on, to, with } [*asset*].”
how turn ⟨ *action*, *asset*, *harm* ⟩ into above format.

- specify **what.** examples (good) 
  "The system shall prevent theft of money"
  "The system shall prevent erasure of account balances"
- not **how** (that’s for requirements & countermeasures). examples (bad) 
  "the system shall use encryption to prevent reading of messages"
  "the system shall use authentication to verify user identities”, "the system shall resist attack”

in-class exercise: security goals for the Grade Management System (GMS)

Examples

format "The system shall { prevent, detect } [*action*] { on, to, with } [*asset*]."
how turn ⟨ *action*, *asset*, *harm* ⟩ into above format.

- specify **what.** examples (good) 
  "The system shall prevent theft of money"
  "The system shall prevent erasure of account balances"
- not **how** (that's for requirements & countermeasures). examples (bad) 
  "the system shall use encryption to prevent reading of messages"
  "the system shall use authentication to verify user identities", "the system shall resist attack"

"The system shall prevent disclosure of grade (by those unprivileged to see it)"
"The system shall prevent overwriting of grade (by those unprivileged to do so)"
"The system shall detect spamming of requests of grade."

feasibility analysis

Compromise

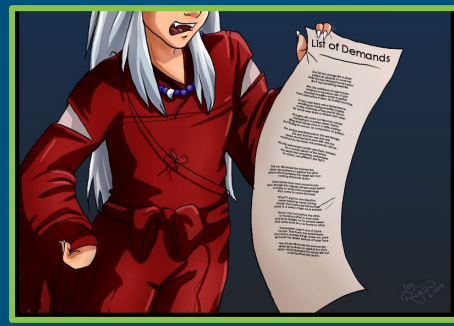
not all goals are feasible to achieve

- impossible
- impractical
- too expensive

relax goals:

- "prevent theft of items from a vault", to
- "resist penetration for 30 minutes", or to
- "detect theft of items from a vault"

security requirements



From Goals to Requirements

why not satisfied with security goals?

goals: what should never happen in any situation ← not testable

requirement: what should happen in specific situations ← testable!

security requirements: constraint on *functional requirements*, in service of *security goals*.

Goals vs. Requirements

goals	requirements
broad scope	narrow scope
apply to system	apply to individual functional requirements
state desires	state constraints
not testable	testable
not about design/implementation details	provide some details

Examples

security requirements: constraint on *functional requirements*, in service of *security goals*.

example

functional requirement: allow people to cash checks
security goal: prevent loss of revenue through bad checks
security requirement: check must be drawn on bank where it's being cashed (so funds can be verified), or customer must be account holder at bank & depositing funds in account (so funds could be reversed)

example

functional requirement: allow two users to chat using IM
security goal: prevent disclosure of message contents to other users
security requirement: contents of message cannot be read by anyone other than the two users
security requirement: message is encrypted by key shared with the two users
- better; doesn't over-commit to encryption algorithm, key size, etc.

Security Requirements

Exercise

security goal: "The system shall { prevent, detect } [*action*] { on, to, with } [*asset*]."
security requirements: constraint on *functional requirements*, in service of *security goals*.

In-class:

functional requirements: students view grades, profs view and change grades,
admins manage enrollment

security goals: "The system shall prevent disclosure of grade (by those unprivileged to see it)"
"The system shall prevent overwriting of grade (by those unprivileged to do so)"
"The system shall detect spamming of requests of grade."

security requirements: *combine functional requirements with goals to invent constraints on system*

Security Requirements

Exercise

security goal: "The system shall { prevent, detect } [*action*] { on, to, with } [*asset*]."
security requirements: constraint on *functional requirements*, in service of *security goals*.

In-class:

functional requirements: students view grades, profs view and change grades,
admins manage enrollment

security goals: "The system shall prevent disclosure of grade (by those unprivileged to see it)"
"The system shall prevent overwriting of grade (by those unprivileged to do so)"
"The system shall detect spamming of requests of grade."

security requirements: grade can only be read by professor, and student it belongs to.
grade can only be written by professor.
spamming of requests for grades must be logged and notified to admins.

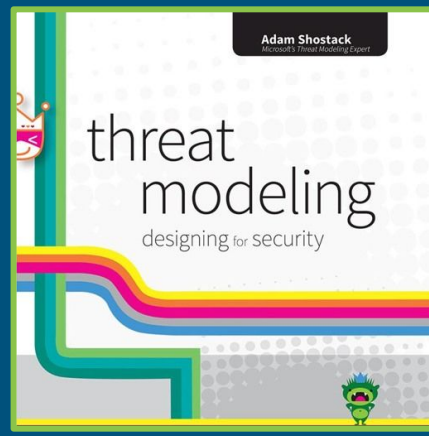
Summary

methodology to arrive at **security requirements**.

1. functional requirements
2. threat analysis
3. harm analysis
4. security goals
5. feasibility analysis
6. security requirements

when do to this: from **beginning** of project (!!!)
security should be **at the core** of your design.
this is **threat modeling**. some steps hard (e.g. step 3). existing methodologies?

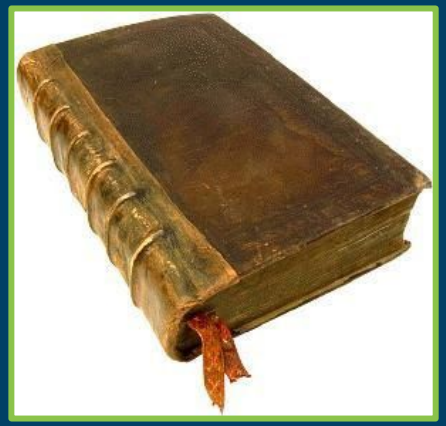
threat modeling



Let's Look at Older Approaches

we've seen **asset- and attacker-centric** threat modeling

- during software development,
- operate on user stories, modify functional requirements



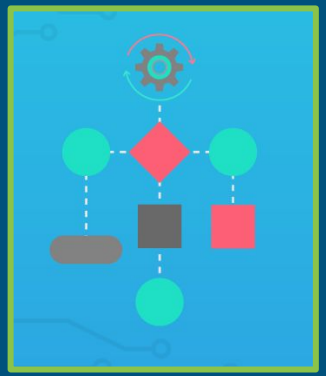
system-centric: understand threats to an existing system

- recon
- predict
- categorize

spoiler: some useful ideas... but not compatible w/ modern SW development.

recon:

Diagrams



start with diagrams, to understand

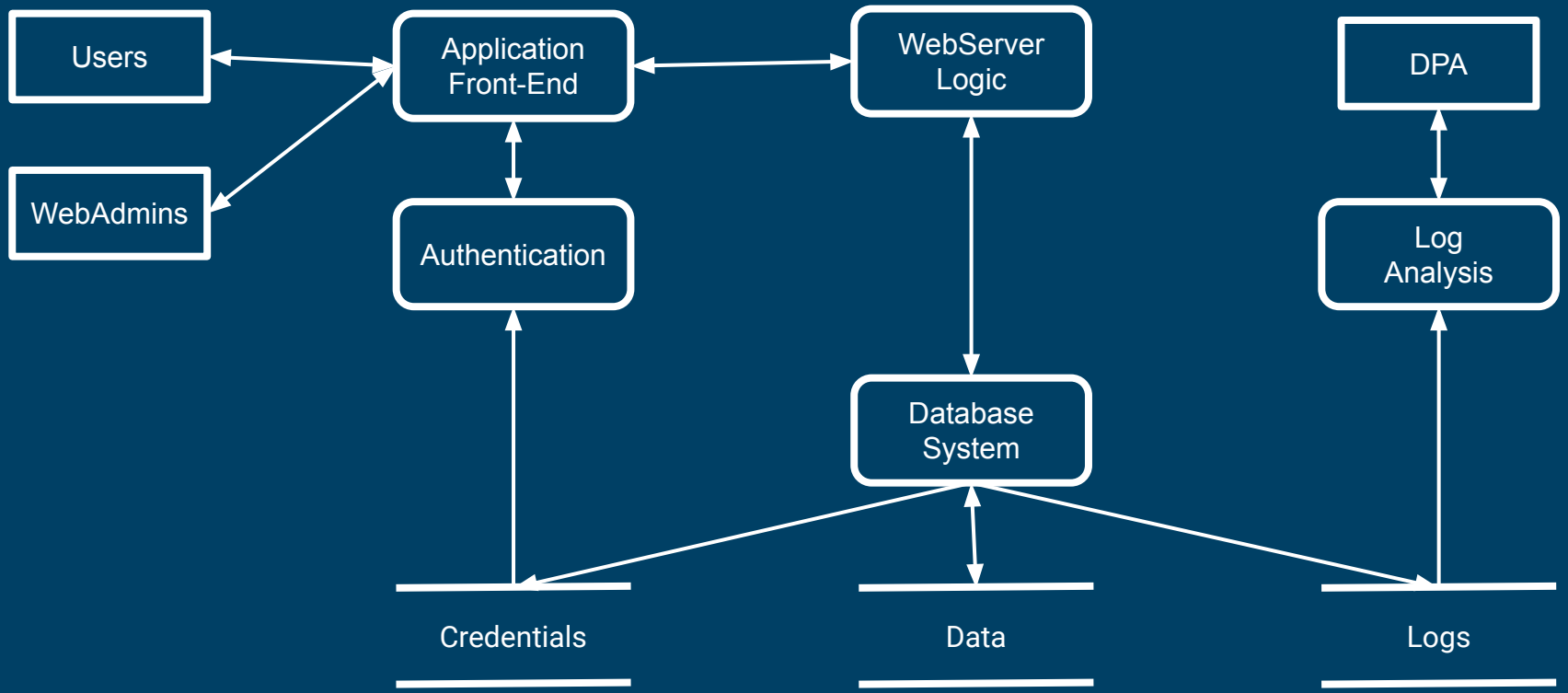
- where data flows in and out
- where data is processed
- **trust boundaries**

good diagrams for this:

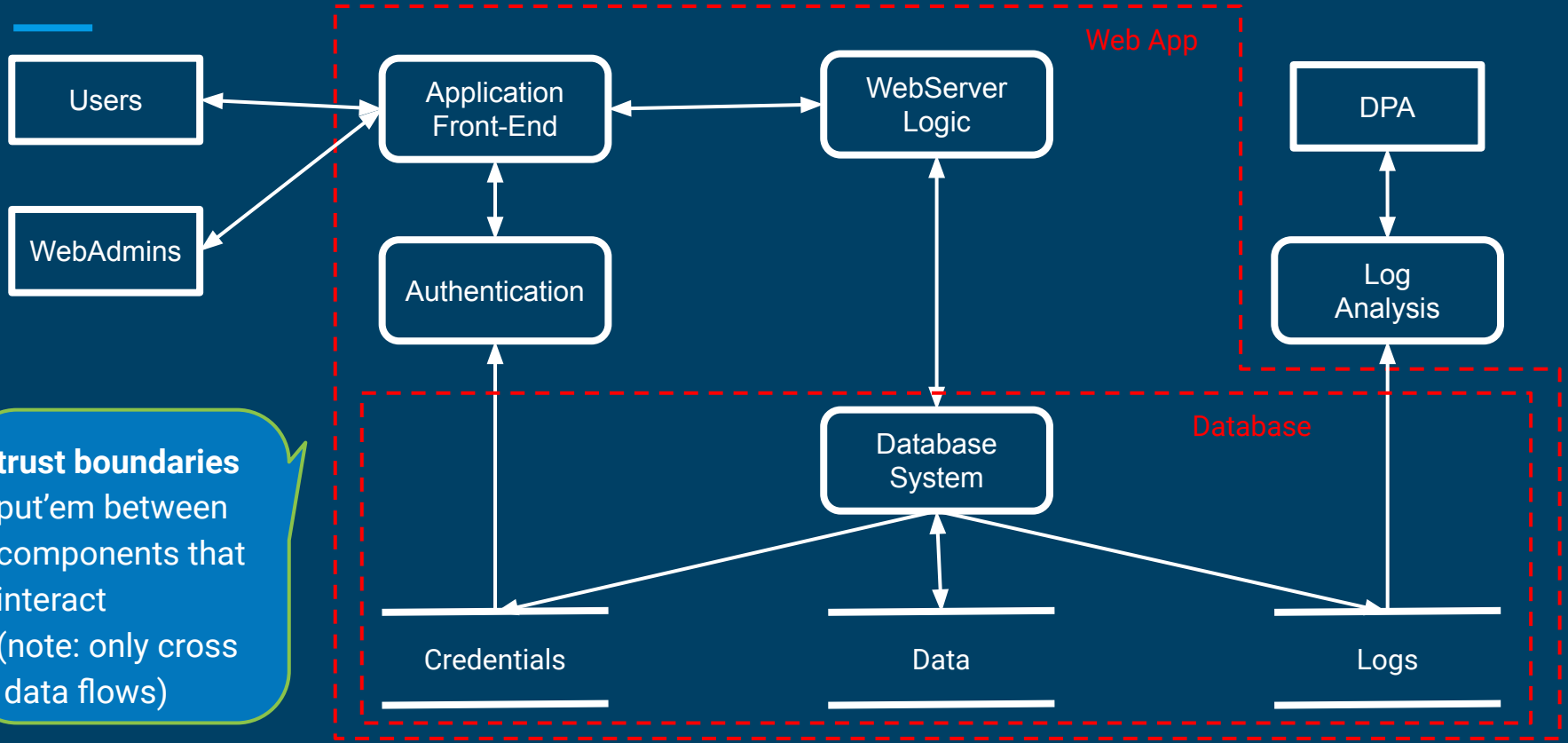
- UML diagrams (structural & behavioral)
- Data Flow Diagrams

Data Flow Diagram (DFD) (web application)

database performance analyzer



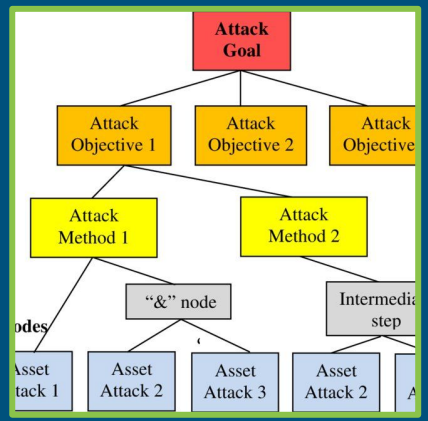
Data Flow Diagram (DFD) (web application)



trust boundaries put'em between components that interact
(note: only cross data flows)

predict:

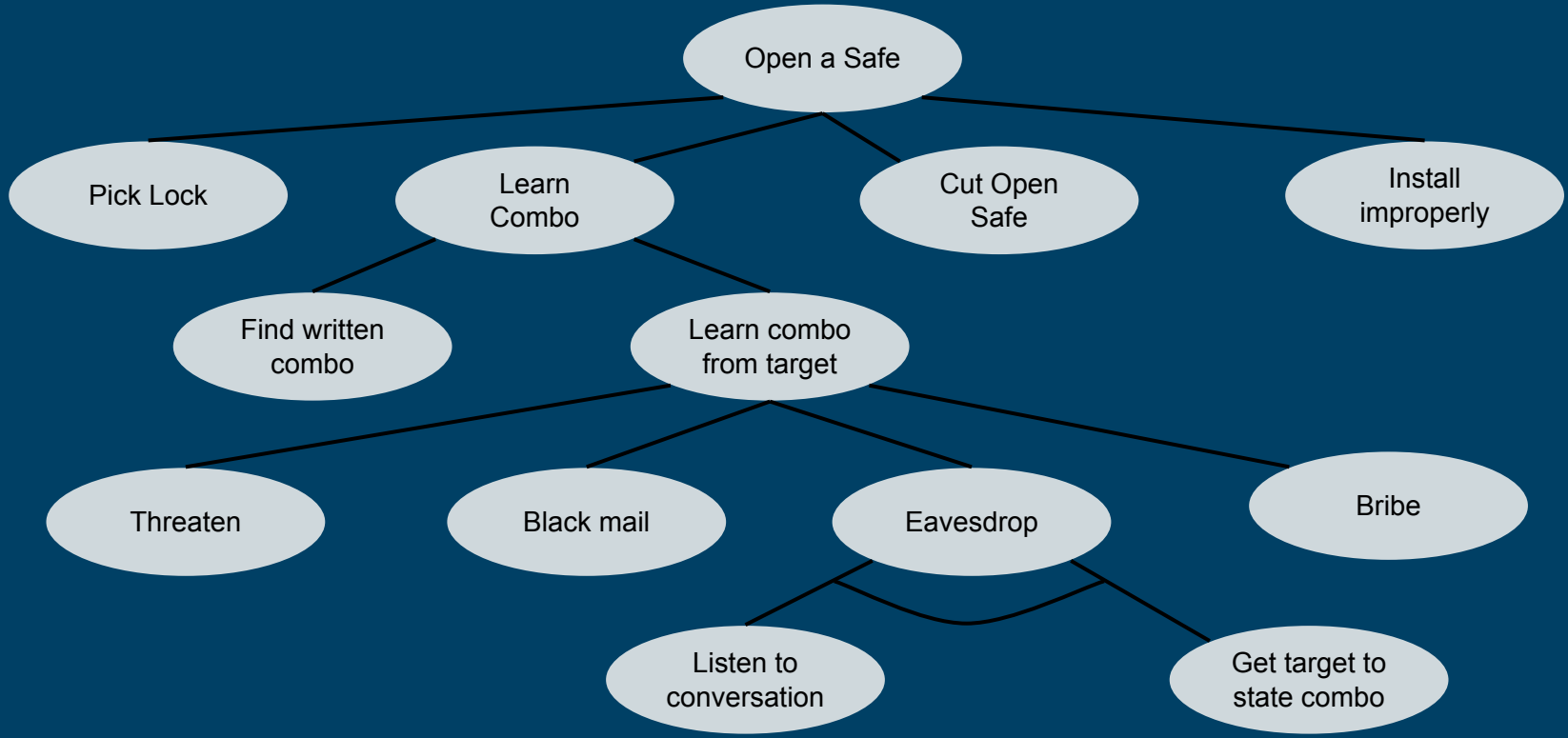
Attack Trees



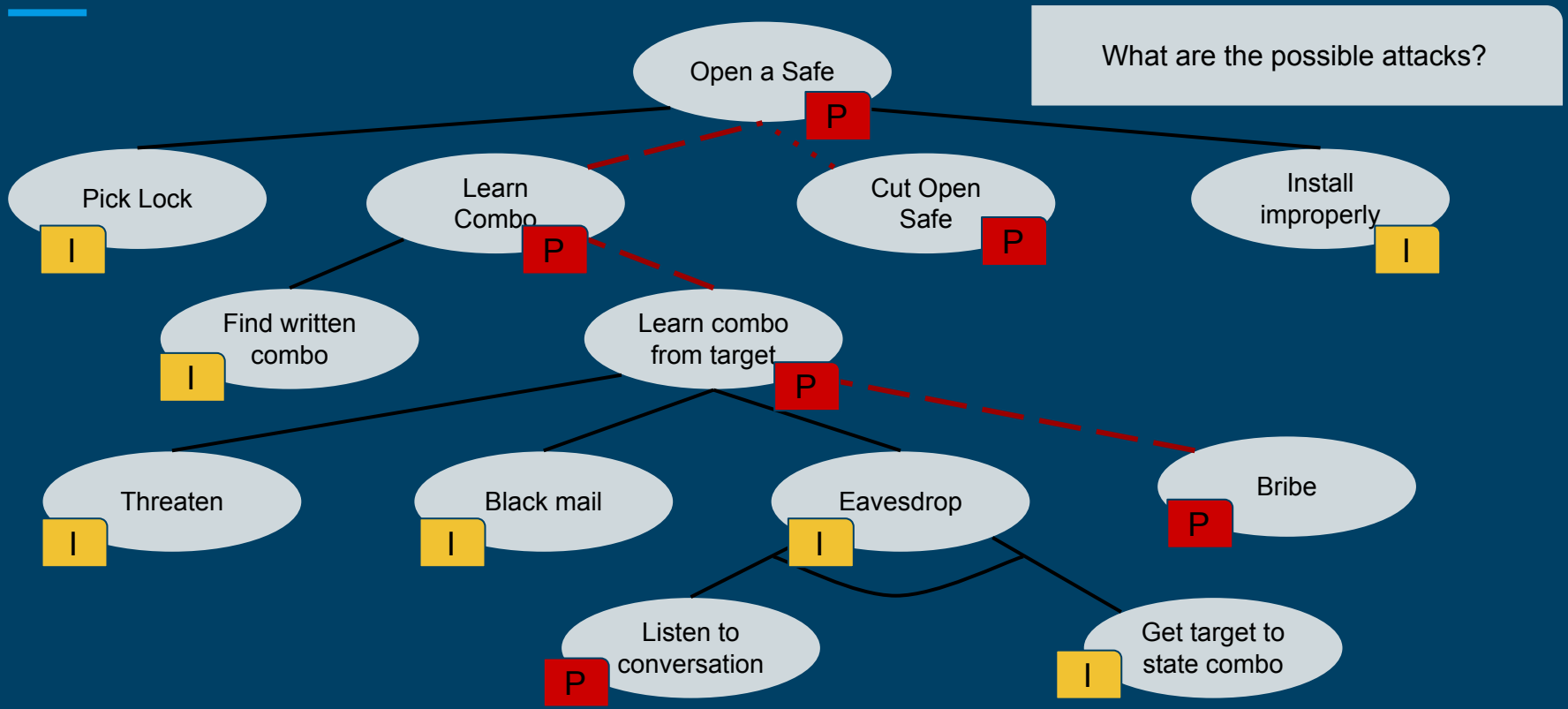
then, conjure up attack scenarios.
structured approach: **attack trees**.

- nodes represent attacks
  - root-node: global goal of an attacker
  - child-node: refinements of this goal
  - leaf-node: un-refineable attack
- child connectives
  - OR: possible refinements
  - AND: necessary refinements

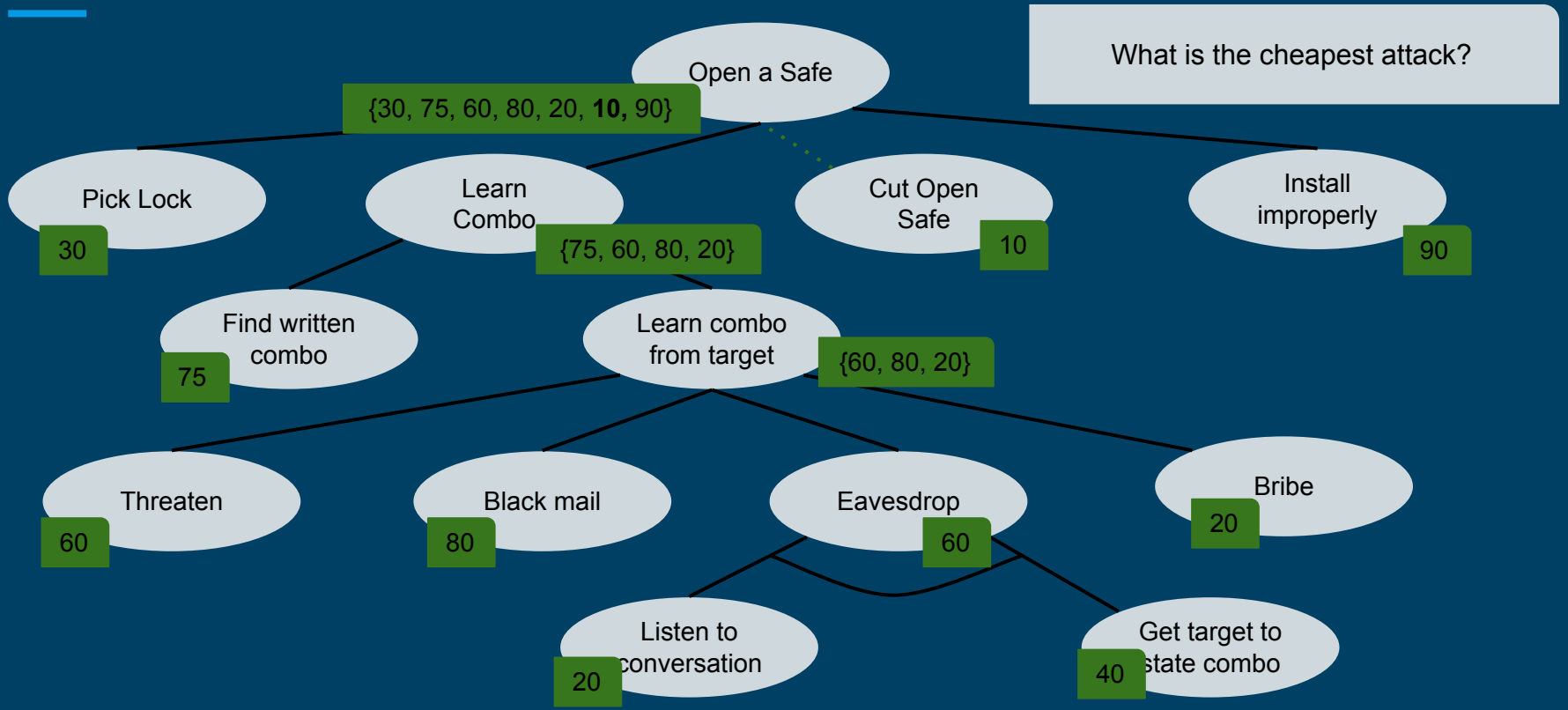
Attack Tree, Example



Attack Tree, Example, Annotate w/ Attributes



Attack Tree, Example, Annotate w/ Attributes



Further Tips

Textual Representation

Goal: Open Safe

1. Pick lock (OR)
2. Learn combo (OR)
    - 2.1. Find written combo (OR)
    - 2.2. Get combo from target (OR)
        - 2.2.1. Threaten (OR)
        - 2.2.2. Black mail (OR)
        - 2.2.3. Eavesdrop (OR)
            - 2.2.3.1. Listen to conversation (AND)
            - 2.2.3.2. Get target to state combo
        - 2.2.4. Bribe
3. Cut Open Safe (OR)
4. Install Improperly

How To Make Attack Trees

- identify possible goals
    - each goal forms separate tree, rooted in higher goal
- continue iterating until you reach all leaves
    - good to involve lots of people
- trees can be reused, as part of larger tree
    - compartmentalization

Theory, Tools

There's a theory, and tools (ADTool)

categorize:

STRIDE

finally, categorize the attacks

STRIDE: a way to categorize attacks. (that's all it is). categories:

- spoofing
- tampering
- repudiation
- information disclosure
- denial of service
- elevation of privilege

decent overview of possible attacks. informs design of countermeasures.

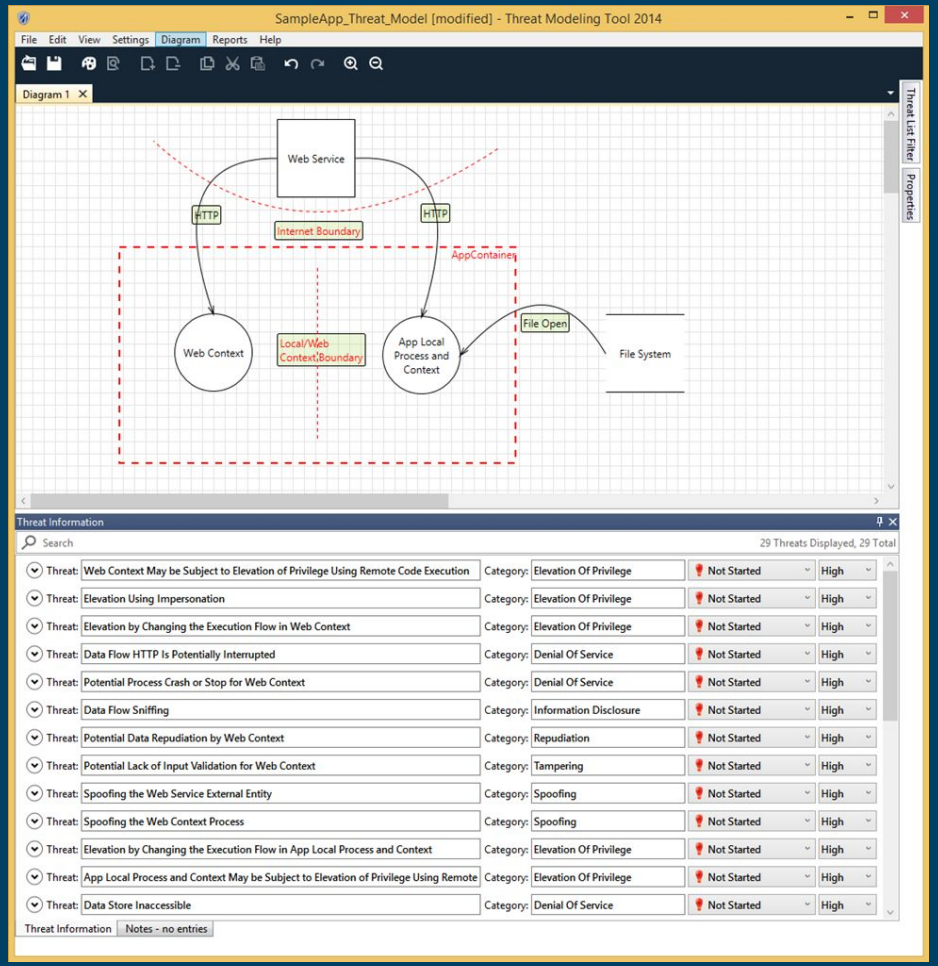
Closing

when to stop threat modeling:

- attack per category
- attack per element in model

Lots of opportunity to re-use kinds of attacks...

Microsoft Threat Modeling Tool



Criticism

this threat modeling approach is from the early 2000s.

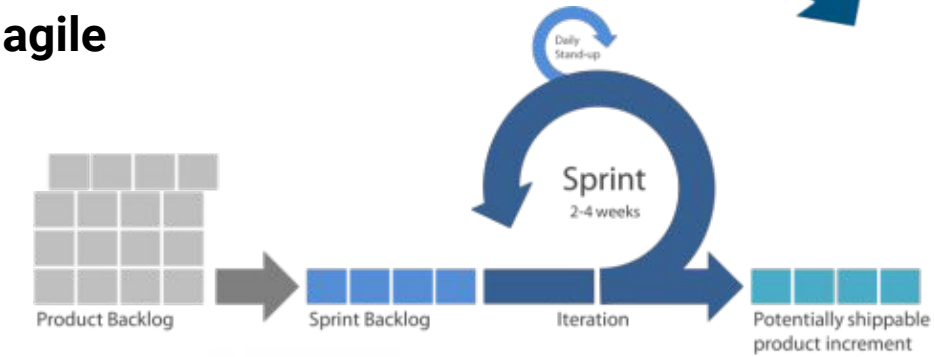
software development has transformed a lot since then.

push to deliver releases faster (CI/CD).
security considered a **hurdle/hindrance**.
diagrams are left out.

creating diagrams is **costly**.
business people don't understand them.

waterfall

agile



DevOps

Criticism - What to do?

Threat Modeling in DevOps?

focus on **user stories**.
understand things in terms of business **assets**.

what makes a difference to a business?

- private data
- financial assets
- critical functions
- people assets
- trade secrets

threats

- theft
- fraud
- exposed data
- interrupted business

What Developer Does

implement three types of controls, per threat:

detection controls enable your code in that user story to detect that type of threat. (e.g. logging)

mitigation controls make attacks slower & more difficult, so I have time to react to it.

defense controls shut down the attack (e.g. disable user account).

Diagrams? Attack trees? STRIDE? All gone.

Criticism

“there is no single best or correct way of performing threat modeling, it is a question of trade-offs and what we want to achieve by doing it”

Source: A. Shostack, “Experiences Threat Modeling at Microsoft,” in Modeling Security Workshop, in Association with MODELS’08, 2008.

Evaluation

Assurance

- How do you convince yourself that system is secure?
- How do you convince others??
- Assurance is evidence that system will not fail in particular ways
  - Development process (e.g. formal methods, deliberate fault injection and discovery)
  - Skill of developers
  - Experience with deployed system
- Evaluation is process of establishing assurance
  - developers
  - QA teams
  - third party labs

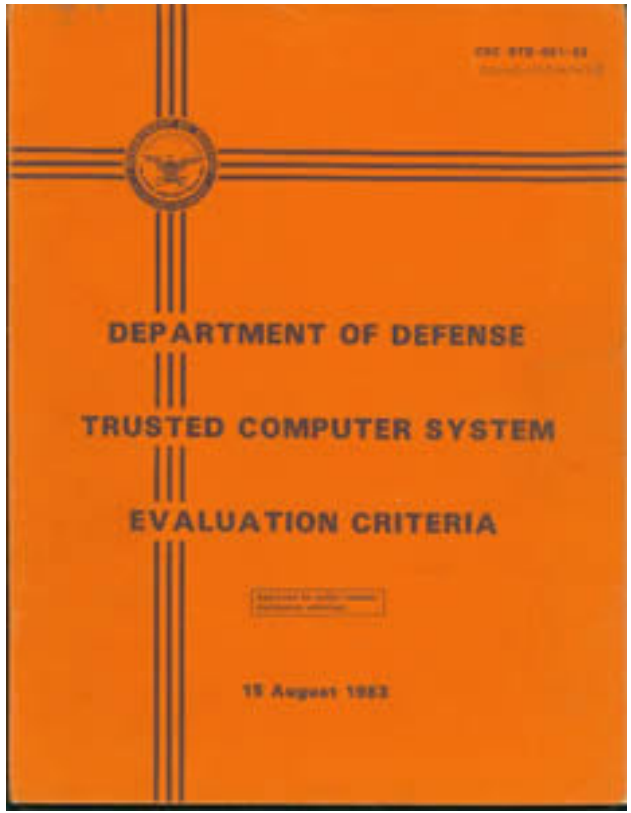
Economics is against us

- Companies race to ship innovative product sooner than competitors
  - Little security
  - Wrong security
- Later security is "bolted on" as additional features
  - But incentive is to lock in customers
  - Product is already deployed; too late for major design changes that might be necessary

Build security in

- Integrate security functionality from the beginning of development
  - During requirements engineering
  - During system design
  - During testing
- Accumulate evidence of security as development proceeds
  - Documentation
  - Analysis: by humans, by machines
  - Test suites

Orange Book evaluation



Orange Book evaluation

- Used approx. 1985-2000 for US government systems
- http://csrc.nist.gov/publications/history/dod85.pdf
- Evaluation classes (selected traits):
  - **D:** meets no higher requirements
  - **C1:** DAC & authentication (but maybe not at the level of individual users), TCB with integrity verification, security testing, documentation of security features/testing/design
  - **C2:** improved DAC (at the level of single users, failsafe defaults, limits on propagation), audit (of specified security relevant events and details of those events)
    - IBM mainframes and Windows NT got this certification

Orange Book evaluation

- Evaluation classes, continued:
  - **B1:** informal security policies, mandatory access control (multilevel security)
  - **B2:** formal security policies, clearly defined TCB, covert channel analysis
  - **B3:** minimal TCB with complete mediation, automated intrusion detection
  - **A1:** formal verification of design
    - only a handful of systems ever achieved this level

Legacy of Orange Book

- Evaluation didn't succeed in commercial market
  - Too costly; costs diverted to government and customers
  - Too long to get evaluated (>1 year) compared to short product cycles
- Raised awareness of security for vendors and government
  - Major operating systems did incorporate discretionary access control; would that have happened without evaluation?
  - But few systems ever incorporated the multilevel security the US DoD wanted
- Unpopular security features mandated by higher levels
  - Research still ongoing on how to make such features usable
- Led to international standards for evaluation…

Common Criteria (CC)

- Evolved in the 1990s out of criteria in Europe, Canada, and US
- Different evaluation model:
  - Define *protection profile* and *security target*
    - think of these as customized security goals/requirements
    - e.g., for OS, for smartphone, for VPN client
    - not one-size-fits-all like Orange Book
  - Increasingly strict evaluation criteria for how well system meets profile/target
- Evaluation done by independent labs

Protection profile (PP)

- Written for a category of products or systems that meet specific consumer needs
- Implementation independent
- Security environment:
  - assumptions about intended usage
  - threats of concern
- Security goals and requirements [using our terminology]
  - Hundreds of pages of pre-written proto-requirements: http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R2.pdf
- PP itself can be evaluated (complete, consistent, technically sound)

Security target (ST)

- Can be based on multiple protection profiles, or created from scratch
- Customized to a specific Target of Evaluation (TOE), i.e., product or system
- Argues (provides evidence) how the system meets the security goals and requirements
  - Assurance argument

Evaluation Assurance Level (EAL)

- **EAL1:** Functionally Tested
  - Analysis of specifications, documentation; independent testing
  - Some confidence desired but threat is not serious
- **EAL2:** Structurally Tested
  - Analysis also of high-level design, of developer's testing; vulnerability analysis
  - Low level of assurance, perhaps for legacy systems
- **EAL3:** Methodically Tested and Checked
  - Also requires use of development environment controls and configuration management

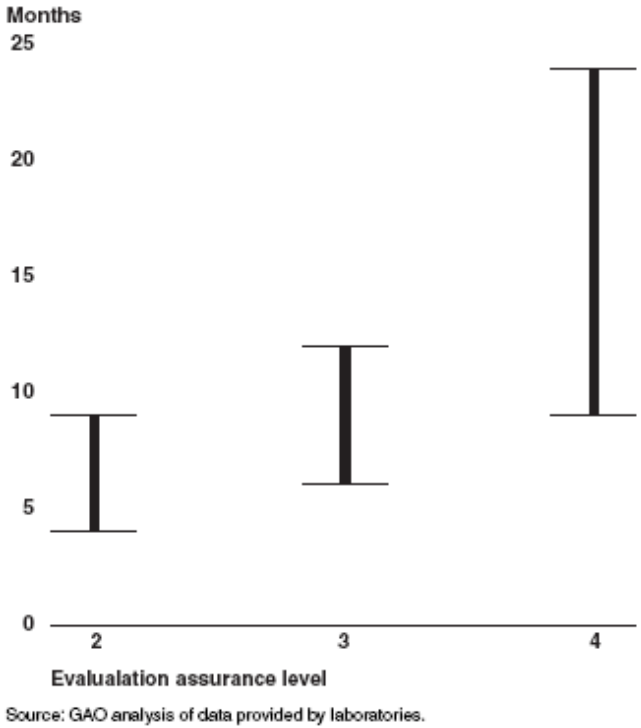
Evaluation Assurance Level (EAL)

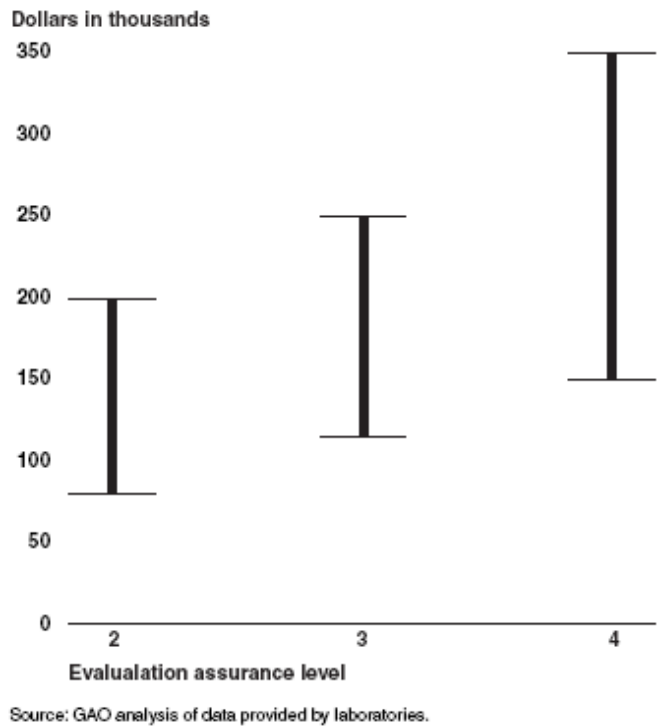
- **EAL4:** Methodically Designed, Tested, and Reviewed
  - Also analyze low-level design, some of the implementation; developers must provide informal model of product or security policy
  - Moderate level of assurance, probably highest likely to achieve for pre-existing systems
  - Common level for commercial OS
- **EAL5 through EAL 7**
  - Increasing demands for formal verification, penetration testing, independent testing
- Higher EAL does not mean more secure—rather, means assurance in claimed security is based on stronger evidence

Legacy of Common Criteria

- "When presented with a security product, you must always consider whether the salesman is lying or mistaken." – Ross Anderson
- Is the PP really what you want?
- Is the evaluation facility trustworthy?
  - Paid by developer
  - Controlled by governments
- What vulnerabilities have been discovered after evaluation?

Evaluation Assurance Level (EAL)





Source: US government report GAO-06-392, 2006

Summary

Summary

Security Engineering - A Lost Cause?

now you know how **Security Engineers** operate:

- understand security built-in is important
  - and want it, too
- valuable lessons on writing secure software
  - principles
- have an idea of what mechanisms exist
  - to give assurance
- have an idea of standard security expectations
  - evaluation

mechanisms

accountability

we *do not know* how to do security engineering.
we do our best w/ what we've got, to make attacks **unaffordable** or too **risky**.